A Derivative-Free Algorithm for Linearly Constrained Optimization Problems

By

Elzain Ahmed Elzain Gumma (elzain@aims.ac.za)

A thesis submitted for the degree of
Doctor of Philosophy

Supervised by:

Dr. Mohsin Hassan Hashim

Faculty of Mathematical Sciences
University of Khartoum, Sudan.

23 September 2010

Acknowledgements

It is a pleasure to acknowledge the support and encrouragement that I have received during my thesis research. Firstly, I would like to thank my supervisor Dr. Mohsin Hassan Hashim for guiding me through the study, he is always available when I needed his advise. I am deeply indebted to Professor M. J. D. Powell, an emeritus Professor at the centre for mathematical sciences, University of Cambridge, for his help. He responded to all my questions and emails. Also I am grateful to Dr. Eihab Bashier, because I received a lot of valuable comments from him. There are many people whom I owe thanks, specially Dr. Eltayeb Abdellatif for being with me all the way and Dr. Jamal Elabawabi for giving me the key to this field.

Parts of my thesis research were made during my two visits to African Institute for Mathematical Sciences (AIMS), South Africa. I am very grateful to all AIMS staff. Special thanks to Professor Fritz Hanhe and Professor Barry Green the previous and current directors of AIMS, respectively, for their excellent hospitality and facilites that supported the research. Also my thanks are extended to Professor Montaz Ali, Ms. Gudrun and Ms. Fransis for their help. During my study, the Nile centre for technology research supported my airtecket to South Africa, I would like to thank them for this help.

I would like to express my gratitude to my colleagues and bosses of Faculty of Pure and Applied Sciences, International University of Africa. I would like to thank them for their patience and support, special thanks to Ustaz Abdella Salah Hamid, Prefessor Ali Ahmed Mahmoud and Dr. Ahmed Elnubi. Finally, I would like to thank all staff of the Faculty of mathematical sciences, University of Khartoum. Special thanks to Dr. Manar, the Dean of the Faculty, for the facilites that suport the research.

Abstract

Derivative-free optimization is an active area of research, because there are many practical problems for which the derivatives are not available, and it may still be desirable to carry out optimization. The main motivation for the study of such problems is the high demand for the solution for such problems.

In this thesis a new derivative-free algorithm has been developed, named LCOBYQA. The main aim of this algorithm is to find a minimum $\underline{x}^\star \in \mathbb{R}^n$ of a nonlinear objective subject to linearly inequality constraints.

The algorithm is based on the trust region method, and uses well known techniques such as the active set version of truncated conjugate gradient method, multivariate Lagrange polynomial interpolation, and QR factorization.

Each iteration of the algorithm constructs a quadratic approximation (model) of the objective function that satisfies interpolation conditions and leaves some freedom in the model, taken up by minimizing the Frobenius norm of the change of the second derivative of the model. A typical iteration of the algorithm generates a new vector of variables either by minimizing the quadratic model subject to the given constraints and the trust region bound, or by a procedure that should improve the accuracy of the model.

Numerical results show that LCOBYQA works well and is so competing against available model-based derivative-free algorithms, such as CONDOR, COBYLA, UOBYQA, NEWUOA and DFO. Under certain conditions LCOBYQA is observed to work extremmely and amazingly fast, leaving an open further investigation to be considered.

المستخلص

تعتبر الأمثلية الخالية من المشتقة من المجالات النشطة في البحث العلمي، ذلك لوجود العديد من المسائل العملية التي يتعذر حساب مشتقاتها. الحاجة الماسة لحل هذا النوع من المسائل كانت الدافع الرئيسي لدراستنا لها.

لقد تم في هذا البحث بناء خوارزمية جديدة سميت LCOBYQA\ ، لحل مسائل الأمثلية دون حساب المشتقة. تهدف الخوارزمية لايجاد الحل الأمثل دون استخدام المشتقة للمسائل غير الخطية ذوات القيود الخطية.

ترتكز الخوارزمية على طريقة منطقة الثقة كما تستخدم طرائق عدة مثل طريقة القيود النشطة، طريقة الاستكمال للدوال متعددة المتغيرات و طريقة عوامل QR\ . كل تكرار في الخوارزمية اعلاه يقوم ببناء تقريب (نموذج) تربيعي لدالة الهدف محققًا شروط تعطي بعض الحرية في النموذج يتم تكملتها بتصغير مقياس فروبنيوس للمشتقة الثانية للنموذج.

أظهرت النتائج العددية أن الخوارزمية تعمل بصورة جيدة و منافسة للخوارزميات المستخدمة حاليًا مثل CONDOR\ ، COBYLA\ ، UOBYQA\ ، NEWUOA\ ، DFO\ و تحت ظروف معينة لوحظ أن الخوارزمية تعمل بصورة مذهلة وسريعة مما يفتح المجال لدراسة الامر و النظر فيه مستقبلًا.

DEDICATION

I dedicate this effort to:

my parents, small family, brothers and sisters.

Contents

Chapter 1

Introduction

Generally, in optimization problems, there is useful information in the derivative of the function one wishes to optimize. For instance, the standard characterization for any local minimum is that the gradient of the objective function is zero. However, there are many problems where the derivatives are unavailable or difficult to obtain, and it may still be desirable to carry optimization. Problems of this kind can be solved by approximating the gradient (possibly the Hessian) using finite difference methods or using automatic differentiation (generic name for techniques that use the computational representation of a function to produce an analytic value of the derivative). There are situations where none of these approaches work. For example, if the objective function is computed using a black-box simulation, then automatic differentation does not work. Even though, the finite difference approach is effective in some applications, it cannot be regarded as a general-purpose technique for solving such problems. This is because if the evaluation of th objective function is expensive, the number of function evaluation required can be excessive. Also, the approach can be unreliable in the presence of noise (noise is defined to be an inaccuracy in the function evaluation).

One of the current approaches for solving problems of this type is called *derivative-free optimization methods*. As reported in (Conn et al., 1996; Sheinberg, 2000), derivative-free optimization is the minimization of problems where we assume that the gradient (the

Hessian) of the objective function is not computed for any $\underline{x}$, although we assume that it exists. Derivative-free optimization methods are necessary for such problems, because they do not rely on the derivatives of the function or the derivatives of the constraints. Rather, they build models for the objective function based on sample function values.

The motivations for examing algorithmic solutions of derivative-free optimization methods, is the high demand from practitioner for such tools. As reported in (Conn et al., 2009), there are really a great number of problems in engineering, mathematics, computer science, physics, finance and operation research using derivative free optimization tools. Here are some examples.

- **Automatic error-analysis:** Derivative-free optimization methods have been used for automatic error-analysis, a process in which the computer is used to analyze the accuracy and stabilty of a numerical computation. One example of automatic error-analysis is to analyze how large the growth factor of Gaussian elimination can be for a specific pivoting strategy. Another example is the estimation of matrix condition number and the analysis of numerical stability for fast martix inversion.

- **Engineering design:** A case study in derivative-free optimization is the helicopter rotor-blade design problem. The goal is to find the structural design of a rotor-blade that minimizes the vibration transmitted by the hub.

- **Molecular geometry:** Another area to use derivative-free optimization is to optimize molecular geometry. An example of this would be to consider the geometry of a cluster of N atoms. The aim is to minimize the cluster's total energy.

- **Other applications:** Other applications of derivative-free methods, include, nanotechnology, air-pollution, groundwater problems, medical image registration and dynamic pricing.

Derivative-free optimization methods are not well developed as gradient-based methods. Current algorithms are effective only for unconstrained and simple bound constrained problems. Also all available derivative-free algorithms in the literature were proposed for minor problems. The first constrained model-based derivative-free algorithm was proposed by Powell (Powell, 1994). This algorithm is very slow, because it is based on linear mulivariate interpolation. Other algorithms were also implemented for small problems. The dimension of problems tested on most of these algorithms does not exceed 100 variables. For example, the maximum dimension of problems tested on UOBYQA (Powell, 2002) must be less than 50, on CONDOR (Frank and Bersini, 2005) must be less than 50, and on DFO (Conn, Sheinberg, and Toint, 1998) must be less than 100. The only derivative-free algorithms which dealt successfully with up to 320 variables using 10 work stations, were NEWUOA(Powell, 2006) and BOBYQA (Powell, 2009). In this thesis an algorithm based on the least Frobenius norm method is designed. It extends the work of Powell (NEWUOA, BOBYQA) to solve linearly constrained problems.

1.1 Statement of the problem

In this thesis we consider the optimization problem:

$$\min f(\underline{x}), \quad \underline{x} \in \mathbb{R}^n, \tag{1.1}$$

$$\text{subject to: } A^T\underline{x} \geq \underline{b}, \quad A \in \mathbb{R}^{n\times \dot{m}}, \quad \underline{b} \in \mathbb{R}^{\dot{m}}, \tag{1.2}$$

where $f(\underline{x})$ is a smooth nonlinear real-valued function, and where the gradient and the Hessian of $f(\underline{x})$ are unavailable. The algorithm we are going to design can solve also unconstrained and simple bounds constrained problems. The simple bounds constrained problems

$$\underline{a} \leq \underline{x} \leq \underline{c}, \quad \underline{a}, \ \underline{c} \in \mathbb{R}^n, \tag{1.3}$$

can be written as

$$I\underline{x} \geq \underline{a}, \ -I\underline{x} \geq -\underline{c}. \tag{1.4}$$

So we can set $A = [I, -I]$, $\underline{b} = [\underline{a}, -\underline{c}]^T$, where I is the $n \times n$ identity matrix. In equation (1.3), if we let $\underline{a}$ to be very large negative integer (tends to minus infinity) and $\underline{c}$ to be very large integer (tends to infinity), then we have an unconstrained optimization problem. Thus, our algorithm can solve also unconstrained and simple bound constrained problems.

The main aim of the thesis is to construct an efficient algorithm to solve problem (1.1) subject to the constraint (1.2). The dimension of the problems to be treated is assumed to be relatively large ($n >$100).

Our strategy to solve problems (1.1)-(1.2) is based on approximating the objective function by a quadratic model. This technique is highly useful to obtain a fast rate of convergence in derivative-free optimization, because it contains the curvature of the objective function. Therefore, at each iteration we construct the following quadratic model: $Q(\underline{x} + \underline{p}) = c + \underline{p}^T \underline{g} + \frac{1}{2}\underline{p}^T G \underline{p}$. We cannot define $\underline{g} = \nabla f(\underline{x})$ and $G = \nabla^2 f(\underline{x})$, because as mentioned earlier, the gradient and the Hessian matrix of $f(\underline{x})$ are unavailable. Instead, we determine the constant c, the vector $\underline{g}$ and the symmetric matrix $G \in \mathbb{R}^{n \times n}$ by imposing the interpolation conditions

$$Q(\underline{x}_i) = f(\underline{x}_i), \ i = 1, \ 2, \ldots, s,$$

where $s = \frac{1}{2}(n+1)(n+2)$. In this case, the parameters of Q can be written as a linear system of equations in the coefficients of the model. If we choose the interpolation points so that the linear system is nonsingular, then the model Q will be defined uniquely. On the other hand, the use of a full quadratic model limits the size of the problems that can be solved in practice. One of the methods that overcomes this drawback was proposed by Powell (Powell, 1994). Powell constructed a full quadraric model from fewer data, and uses the the remaining freedom in the model to minimize the Frobenius norm of the second derivative matrix of the change to the model. This variational problem is expressed as a solution of $(m + n + 1) \times (m + n + 1)$ system of linear equations, where m is the number of the interpolation points which satify $n + 2 \leq m \leq \frac{1}{2}(n+1)(n+2)$.

Our algorithm is called LCOBYQA. The name LCOBYQA is acronym for Linearly Con-

strained Optimization BY Quadratic Approximation. LCOBYQA is a Matlab software which is an extension of Powell's algorithms (Powell, 2006, 2009), that solves problem (1.1), subject to the constraint (1.2). LCOBYQA is an iterative algorithm. A typical iteration of the algorithm generates a new vector of variables either by minimizing the quadratic model of the objective function subject to the linear constraint and the trust region bound, or by a procedure that should improve the geometry of the interpolation points.

1.2 Thesis contribution

Our algorithm offers a possibility to solve linearly constrained problems with large dimension without using external libraries and using a single processor.

1.3 Assumptions needed by the software

- No derivatives of $f(\underline{x})$ are required, but the algorithm assumes that they exist.
- The algorithm will only find a local minimum of $f(\underline{x})$.
- It is assumed that the objective function $f(\underline{x})$ might have a limited noise.

1.4 The structure of the thesis

The rest of the thesis is organized as follows. Chapter two presents the background needed for Chapters 3 and 4. It includes an overview of derivative-based methods and model-based derivative-free optimization algorithms. Also, the multivariate interpolation method is presented. Since our algorithm is an extension of Powell's work, the latter is discussed in detail in Chapter 3. This makes it easy to describe our algorithm in Chapter

4. Several numerical results are presented and discussed in Chapter 5. The thesis ends in Chapter 6 with a conclusion and an outlook to possible future research. The proofs of some of the assertions made in Chapter 2 and 3, are given in Appendix A. Finally, codes of the algorithm are given in Appendix B.

Chapter 2

Theoretical Background

2.1 Introduction

In this chapter we present the theoretical background that will be needed in different parts in the thesis. Firstly, some notations and some basic mathematical concepts are introduced in section 2.2. An overview of constrained derivative-based optimization is presented in section 2.3. In section 2.4, we discuss quadratic programming. An active set method is presented in section 2.5. Line search methods and trust-region methods are discussed in section 2.6. The truncated conjugate gradient method is explained in section 2.7. Finally, in section 2.8, we describe the history of model-based derivative-free optimization and an outline of interpolation model-based derivative-free algorithm.

2.2 Vector and Matrix Norms

The most common norms in $\mathbb{R}^n$ are the p-norms (where $p \geq 1$ is a real number) which are given by

$$\| \underline{x} \|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}, \quad 1 \leq p < \infty, \tag{2.1}$$

and the ∞-norm is defined by

$$\| \underline{x} \|_\infty = \max_{1 \leq i \leq n} | x_i | . \tag{2.2}$$

Similarly, we can define the p-norms for matrices. Let A be an $m \times n$ matrix. The most frequently used matrix norms in numerical linear algebra are the p-norms:

$$\| A \|_p = \sup_{\underline{x} \neq \underline{0}} \frac{\| A\underline{x} \|_p}{\| \underline{x} \|_p}, \tag{2.3}$$

and the Frobenius norm

$$\| A \|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}. \tag{2.4}$$

In this thesis, we restrict ourself to the Fobenius norm and the p-norms, where $p = 1, 2$. An important relation between these norms is

$$\sqrt{m} \| A \|_1 \leq \| A \|_2 \leq \| A \|_F \leq \sqrt{n} \| A \|_2 \leq n \| A \|_1 . \tag{2.5}$$

Henceforth, the 2-norm of a vector and matrix will be denoted by $\| . \|$. The following theorem gives the 2-norm of matrices.

Theorem 2.2.1. *Let A be an $m \times n$ matrix, and ω be the largest eigenvalue of $A^T A$. Then $\| A \| = \sqrt{\omega}$.*

Proof: see Appendix A.

2.2.1 Conditioning

Let A be an $n \times n$ nonsingular matrix, the condition number of A is defined by

$$\kappa(A) = \| A \| . \| A^{-1} \| . \tag{2.6}$$

Large values of the condition number usually indicate ill-conditioning. As justification for the previous assertiont, we refere to the following theorem:

Theorem 2.2.2. *Suppose that $\underline{x}$ is the solution of the linear system $A\underline{x} = \underline{b}$, where A is nonsingular matrix, and $\bar{\underline{x}}$ is the solution of the linear system $A\bar{\underline{x}} = \bar{\underline{b}}$, then*

$$\frac{\| \underline{x} - \bar{\underline{x}} \|}{\| \underline{x} \|} \leq \kappa(A).\frac{\| \underline{b} - \bar{\underline{b}} \|}{\| \underline{b} \|}. \tag{2.7}$$

Proof: see (Igor et al., 2009).

2.3 Overview of constrained derivative-based optimization

The structure of most constrained optimization problems is essentially contained in the following. Find

$$\min f(\underline{x}), \quad \underline{x} \in \mathbb{R}^n, \quad \text{subject to:} \quad c_i(\underline{x}) = 0, i \in E, \quad c_i(\underline{x}) \geq 0, i \in \mathsf{I}, \tag{2.8}$$

where E is the index set of the equality constraints, and I is the index set of the inequality constraints. Any point $\underline{x}'$ which satisfies all constraints in (2.8) is said to be a feasible point. The set of all feasible points is called a feasible region. We say that the point $\underline{x}^\star$ is a local minimum of $f(\underline{x})$ if, $f(\underline{x}^\star) \leq f(\underline{x})$, for all feasible $\underline{x}$ in the neighbourhood of $\underline{x}^\star$. A constraint c_i is said to be active at $\underline{x}'$ if $c(\underline{x}') = 0$. This means that all the equality constraints are active. The set of active constraints at the solution of (2.8) is very important, because if we know this set, we regard the problem an equality constrained problem, the remaining constraints can be locally ignored. This is because the inactive constraints can be perturbed by small amounts without changing the local solution see (Fletcher, 1987).

2.3.1 Linear constrained optimization

Most of the concepts introduced in this section will be used in different parts of our thesis. There is a class of problems for which the constraints are linear functions.

The general form of the linearly constrained problem is:

$$\min f(\underline{x}), \quad \underline{x} \in \mathbb{R}^n, \quad \text{subject to: } \underline{a}_i^T \underline{x} - b_i = 0, \quad i \in E, \quad \underline{a}_i^T \underline{x} - b_i \geq 0, \quad i \in \mathsf{I}, \ \underline{b} \in R^{\dot{m}}. \tag{2.9}$$

Linear equality constraints

If $\mathsf{I} = \phi$ in equation (2.9), we obtain the linear equality constraints problem which has the general form:

$$\min f(\underline{x}), \quad \underline{x} \in \mathbb{R}^n, \quad \text{subject to: } A^T \underline{x} - \underline{b} = 0, \tag{2.10}$$

where $\mathsf{A} \in \mathbb{R}^{n \times \dot{m}}$ is matrix with a_i as its i-th column. The feasible point $\underline{x}^\star$ is a local minimum of (2.10) only if $f(\underline{x}^\star) \leq f(\underline{x})$ for all $\underline{x}$ in some neighbourhood of $\underline{x}^\star$. In order to derive the optimality conditions for (2.10), we first charactorize the set of feasible points in the neighbourhood of $\underline{x}^\star$. Because the constraints are a linear system, the properties of a linear subspace make it possible to state a simple characterization of all feasible directions from a feasible point. Consider the step between two feasible points $\underline{x}'$, $\underline{x}''$, by linearlity $A^T(\underline{x}' - \underline{x}'') = 0$, since $A^T \underline{x}' = \underline{b}$, $A^T \underline{x}'' = \underline{b}$. Thus, the step $\underline{p}$ from a feasible point to any other feasible point must satisfy:

$$A^T \underline{p} = \underline{0}. \tag{2.11}$$

The step $\underline{p}$ that satisfes (2.11) is called a feasible direction with respect to the equality constrained problem. Any step from a feasible point $\underline{x}'$ along such a direction does not violate the constraints, since $A^T(\underline{x}' + \alpha \underline{p}) = A^T \underline{x}' = \underline{b}$. Therefore, equation (2.11) completely characterizes feasible perturbations. The set of vectors $\underline{p}$ satisfying (2.11) is called the null space of A^T, which is a linear space.

Let the columns of a matrix Z form a basis for the null space of A^T, then $A^T Z = O$, and every feasible direction can be written as a linear combination of the columns of Z. Therefore, any $\underline{p}$ satisfying (2.11) can be written as $\underline{p} = Z \underline{p}_z$. In order to determine the optimality conditions of a given feasible point $\underline{x}^\star$, we examine the Taylor expansion of $f(\underline{x})$ around $\underline{x}^\star$ along a feasible direction $\underline{p} = Z \underline{p}_z$. Thus,

$$f(\underline{x}^\star + \alpha \underline{p}) = f(\underline{x}^\star + \alpha Z \underline{p}_z) = f(\underline{x}^\star) + \alpha \underline{p}_z^T Z^T \underline{g}(\underline{x}^\star) + \frac{1}{2} \alpha^2 \underline{p}_z^T Z^T G(\underline{x}^\star + \alpha \theta \underline{p}) Z \underline{p}_z, \tag{2.12}$$

where θ satisfies $0 \leq \theta \leq 1$, $\underline{g}(\underline{x}) = \nabla f(\underline{x}),\ G = \nabla^2 f(\underline{x})$ and α is taken, without loss of generality, as a positive scalar. If $\underline{p}_z^T Z^T \underline{g}(\underline{x}^\star)$ is negative, then there is a decent direction from $\underline{x}^\star$, also if $\underline{p}_z^T Z^T \underline{g}(\underline{x}^\star)$ is positive, then -$\underline{p}_z^T Z^T \underline{g}(\underline{x}^\star)$ is negative, so there is a decent direction from $\underline{x}^\star$, thus a necessary condition for $\underline{x}^\star$ to be a local minimizer is that

$$Z^T \underline{g}(\underline{x}^\star) = \underline{0}. \tag{2.13}$$

The vector $Z^T \underline{g}(\underline{x}^\star)$ is called the projected gradient of $f(\underline{x})$ at $\underline{x}^\star$. Any point at which the projected gradient vanishes is called a constrained stationary point. The result (2.13) implies that $\underline{g}(\underline{x}^\star)$ must be a linear combination of the columns of A, i.e.

$$\underline{g}(\underline{x}^\star) = \sum_{i=1}^{\dot{m}} a_i \lambda_i^\star = A\underline{\lambda}^\star, \tag{2.14}$$

for some vector $\underline{\lambda}^\star$, which is called the vector of Lagrange multipliers. The vector $\underline{\lambda}^\star$ is unique only if the columns of A are linearly independent. By subsitituting (2.13) in (2.12), we have

$$f(\underline{x}^\star + \alpha Z \underline{p}_z) = f(\underline{x}^\star) + \frac{1}{2}\alpha^2 \underline{p}_z^T Z^T G(\underline{x}^\star + \alpha\theta\underline{p}) Z\underline{p}_z. \tag{2.15}$$

If $Z^T G Z$ is negative definite, then this contradicts that $\underline{x}^\star$ is a local minimizer, thus $Z^T G Z$ must be positive semi-definite. So the necessary conditions for $\underline{x}^\star$ to be a local minimizer for (2.10) are the following:

- $A^T \underline{x}^\star = \underline{b}$
- $Z^T \underline{g}(\underline{x}^\star) = \underline{0}$, or, equivalently $\underline{g}(\underline{x}^\star) = A\underline{\lambda}^\star$
- $Z^T G Z$ is positive semi-definite.

Sufficient conditions for $\underline{x}^\star$ to be a local minimizer for (2.10) are:

- $A^T \underline{x}^\star = \underline{b}$
- $Z^T \underline{g}(\underline{x}^\star) = \underline{0}$, or, equivalently $\underline{g}(\underline{x}^\star) = A\underline{\lambda}^\star$
- $Z^T G Z$ is positive definite.

Linear inequality constraints

If $E = \phi$ in equation (2.9), we obtain the linear inequality constrained problem:

$$\min f(\underline{x}), \quad \underline{x} \in \mathbb{R}^n, \quad \text{subject to: } A^T\underline{x} - \underline{b} \geq 0. \tag{2.16}$$

We try to derive the characterization of feasible points in the neighbourhood of a feasible solution. To do so, it will be important to distinguish between the active and inactive constraints at a feasible point $\underline{x}'$. The active constraints have a special significance because they restrict the feasible perturbations about a feasible point. If the j-th constraint is inactive at a feasible point $\underline{x}'$, then it is possible to move a non-zero distance from $\underline{x}'$ in any direction without violating that constraint, i.e., for any vector $\underline{p}$, $\underline{x}' + \alpha\underline{p}$ will be feasible with respect to an inactive constraint if $\mid \alpha \mid$ is sufficiently small. On the other hand, an active constraint restricts feasible perturbations in every neighbourhood of a feasible point. Suppose that the i-th constraint is active at $\underline{x}'$, so that $\underline{a}_i^T\underline{x}' = \underline{b}_i$. There are two categories of feasible directions with respect to an active constraint.

Firstly, if $\underline{p}$ satistfies $\underline{a}_i^T\underline{p} = 0$. In this case the direction is called a binding perturbation with respect to the i-th constraint. In the case of binding direction, i-th constraint remains active at all points $\underline{x}' + \alpha\underline{p}$ for any α.

Secondly, if $\underline{p}$ satisfies $\underline{a}_i^T\underline{p} > 0$. In this case $\underline{p}$ is called a non-binding perturbation with respect to the i-th constraint. Since $\underline{a}_i^T(\underline{x}' + \alpha\underline{p}) = \underline{b}_i + \alpha\underline{a}_i^T\underline{p} > \underline{b}_i,\ \alpha > 0$, the i-th constraint becomes inactive at the perturbed point $\underline{x}' + \alpha\underline{p}$. In order to determine the solution $\underline{x}^\star$ of (2.16), it is necessary to identify the active constraints at $\underline{x}^\star$. Let the $t0$ columns of the matrix $\hat{A}$ contain the coefficients of the constraints that are active at $\underline{x}^\star$, with a similar convention for a vector $\underline{\hat{b}}$, so that $\hat{A}^T\underline{x}^\star = \underline{\hat{b}}$. Let Z be a matrix whose columns form a basis for the set of vectors orthogonal to the columns of $\hat{A}$. Any $\underline{p}$ satisfying $\hat{A}^T\underline{p} = \underline{0}$, can therefore be written as a linear combination of the columns of Z.

The optimality conditions for the linear inequality constraints are given as follows.

The necessary conditions for $\underline{x}^\star$ to be a local minimum of (2.16) are:

- $A^T\underline{x}^\star \geq \underline{b}$, with $\hat{A}^T\underline{x}^\star = \underline{\hat{b}}$
- $Z^T\underline{g}(\underline{x}^\star) = \underline{0}$, or, equivalently $\underline{g}(\underline{x}^\star) = \hat{A}\underline{\lambda}^\star$
- $\lambda_i \geq 0,\ i = 1, 2, \ldots, t0.$

- $Z^T G Z$ is positive semi-definite.

The only complication over the equality constrainred case arise because of the possibility of obtaining a zero Lagrange multiplier corresponding to an active constraint.

Sufficient conditions for $\underline{x}^\star$ to be a local minimizer for (2.16) are:

- $A^T \underline{x}^\star \geq \underline{b}$, with $\hat{A}^T \underline{x}^\star = \hat{\underline{b}}$
- $Z^T \underline{g}(\underline{x}^\star) = \underline{0}$, or, equivalently $\underline{g}(\underline{x}^\star) = \hat{A}\underline{\lambda}^\star$
- $\lambda_i > 0, \ i = 1, 2, \ldots, t0.$
- $Z^T G Z$ is positive definite.

For more details see (Hashim, 1995).

2.4 Quadratic programming

The quadratic programming problem is to minimize a quadratic objective function subject to linear constraints:

$$\min q(\underline{x}) = \underline{g}^T \underline{x} + \frac{1}{2}\underline{x}^T G \underline{x}, \quad \underline{x} \in \mathbb{R}^n, \quad \text{subject to: } \underline{a}_i^T \underline{x} - b_i = 0, i \in E, \quad \underline{a}_i^T \underline{x} - b_i \geq 0, i \in \mathsf{I}, \tag{2.17}$$

where G is symmetric. If $\mathsf{I} = \phi$, we have the simple problem

$$\min q(\underline{x}) = \underline{g}^T \underline{x} + \frac{1}{2}\underline{x}^T G \underline{x}, \quad \underline{x} \in \mathbb{R}^n, \quad \text{subject to: } A^T \underline{x} = \underline{b}. \tag{2.18}$$

It is assumed that there are $\dot{m} \leq n$ constraints. So $A \in \mathbb{R}^{n \times \dot{m}}$, $\underline{b} \in \mathbb{R}^{\dot{m}}$. Assume that A^T has a full rank. This assumption ensures that a vector $\underline{\lambda}$ of unique Lagrange multipliers exists. We consider a generalized elemination method to solve (2.18) see (Fletcher, 1987). Let Y, Z be $n \times \dot{m}$ and $n \times (n - \dot{m})$ matrices, respectively, such that $[Y : Z]$ is nonsingular. In addition, let $A^T Y = I$ and $A^T Z = O$, where I is $n \times n$ identity matrix. Thus, Y^T can be considered as a left inverse of A, so the solution of $A^T \underline{x} = \underline{b}$ is given by $\underline{x} = Y\underline{b}$. However, this solution is not unique. In general any feasible point given by $\underline{x} = Y\underline{b} + Z\underline{y}$, where $\underline{y} \in \mathbb{R}^{n-\dot{m}}$ is any vector, is solution to $A^T \underline{x} = \underline{b}$. The matrix Z has linearly independent columns $Z_1, \ Z_2, \ldots, \ Z_{n-\dot{m}}$ which form a basis for the null space of A^T. At

any point $\underline{x}'$, any feasible correction $\underline{p}$ can be written as

$$\underline{p} = Z\underline{y} = \sum_{i=1}^{n-\dot{m}} Z_i \underline{y}_i. \tag{2.19}$$

Equations (2.19), (2.18), and the equality $\underline{x} = Y\underline{b} + Z\underline{y}$, give:

$$\min q(\underline{y}) = \frac{1}{2}\underline{y}^T Z^T G Z \underline{y} + (\underline{g} + GY\underline{b})^T Z\underline{y} + \frac{1}{2}(\underline{g} + GY\underline{b})^T Y\underline{b}. \tag{2.20}$$

If Z^TGZ is positive definite, then the unique minimizer $\underline{y}^\star$ is computed either using Cholesky factorization (Gill et al., 1988) or using iterative methods. Once $\underline{y}^\star$ is known, then $\underline{x}^\star$ is computed using $\underline{x}^\star = Y\underline{b} + Z\underline{y}^\star$. To obtain $\underline{\lambda}^*$, we premultiply $G\underline{x}^\star + \underline{g} = A\underline{\lambda}^\star$ by Y^T, we get

$$\underline{\lambda}^\star = Y^T(G\underline{x}^\star + \underline{g}). \tag{2.21}$$

Depending on the choice of Y and Z, a number of methods exist for solving (2.18). One choice of a particular importance is obtained by using the QR factorization of A. This can be written as

$$A = Q \begin{bmatrix} R \\ O \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ O \end{bmatrix} = Q_1 R, \tag{2.22}$$

where Q is an $n \times n$ orthogonal matrix and R is $\dot{m} \times \dot{m}$ upper triangular, and the partitions Q_1, Q_2 are $n \times \dot{m}$ and $n \times (n - \dot{m})$ respectively. We choose

$$Y = Q_1 R^{-T} \text{ and } \qquad\qquad Z = Q_2. \tag{2.23}$$

This scheme is due to (Gill et al., 1988). The orthogonal factorization methods is advantageaus in that caculatating Z and Y involve operations with elementry orthogonal matrices which are very stable numerically. Also the choice $Z = Q_2$ gives the best possible bound

$$\kappa(Z^TGZ) \leq \kappa(G) \tag{2.24}$$

on the condition number $\kappa(Z^TGZ)$.

2.5 Active set methods

Active set methods play a vital role in chapter 4 of our thesis, where we use it to solve the quadratic model at each iteration subject to linear constraints.

Consider the problem:

$$\min f(\underline{x}), \quad \underline{x} \in \mathbb{R}^n, \quad \text{subject to: } A^T\underline{x} \geq \underline{b}. \tag{2.25}$$

As the name suggests, an active set method is an iterative method that aims to predict which of the inequality constraints $\underline{a}_i^T\underline{x} \geq b_i,\ i = 1, 2, \ldots, \dot{m}$ are active at the solution of (2.25), because only the active constraints are significient in the optimality conditions, as discussed in section 2.3.
Assume that $t0$ constraints are active at $\underline{x}^\star$, let $\hat{A}^T$ denote the matrix whose i-th row, and $i = 1, 2, \ldots, t0$, contains the coefficients of the i-th active constraint. From the necessary conditions for $\underline{x}^\star$ to be optimal, we have

$$\underline{g}(\underline{x}^\star) = \hat{A}\underline{\lambda}^\star, \quad \underline{\lambda}^\star \geq \underline{0}. \tag{2.26}$$

A crucial distinction from the equality constrained case is the restriction (2.26) on the sign of the Lagrange multipliers. A strictly negative multiplier implies the existance of a feasible descent direction which contradicts the optimality of $\underline{x}^\star$. Thus, the linear inequality constrained problems are inherently more complicated than the equality consrained problem, because the set of active constraints at the solution is generally unknowen. An active set method generates a sequence of feasible points $\underline{x}^1,\ \underline{x}^2, \ldots$, which terminates in a finite number of steps at the the solution $\underline{x}^\star$.
The idea of the active set method arises from the following motivation: If the correct working set at the solution is known a priori, then the solution of the linear inequality problem would be also a solution of the linear equality constrained problem. This make it possible to me to use techniques from the linear equality constrainted problem to solve linear inequality constrained problems. To do so, we select a "working set" of constraints to be treated as equality constraints. An active set method minimizes $f(\underline{x})$ subject to the constraints in the working set. Since the prediction of the active set could be wrong , an

"art" of the active set method is to adjust the working set either by adding a constraint it or by removing those for which further progress is not predicted. For more detials we consider the model algorithm presented in (Gill et al., 1988).

2.5.1 Active set algorithm (Algorithm 2.1)

Let k be the iteration number during a typical active set method. At any iterate $\underline{x}^k$, there is a set of equality constraints associated with the current working set. In particular, $t0_k$ will be the number of constraints in the working set. Let I^k denote the index set of those constraints, $\hat{A}_k^T$ the coefficent matrix of the constraints at $\underline{x}^k$, $\hat{\underline{b}}_k$ the vector of the corresponding right-hand components and Z_k denote the matrix whose sub-columns form a basis for the null-space of $\hat{A}_k^T$. Assuming that we are given a feasible starting point $\underline{x}^0$, set $k = 0$, determine $\underline{t0}_0$, I^0, $\hat{A}_0^T$, $\hat{\underline{b}}_0$, and execute the following steps:

- **Step 1:** [Test of convergence]. If the convergence condition is satisfied at $\underline{x}^k$, the algorithm terimates at the solution $\underline{x}^k$.
- **Step 2:** [Choose which logic to perform] decide whether to continue minimizing in a current subspace or whether to delete a constraint from the working set. If a constraint is to be deleted, then go to step 6.
- **Step 3:** [Compute a feasible direction]: compute a non-zero $(n\text{-}t0_k)$-vector $\underline{p}_z$ (see section 2.5.2). Then the search direction $\underline{p}_k$ is given by $\underline{p}_k = Z_k \underline{p}_z$
- **Step 4:** [Compute the step length]: Compute $\hat{\alpha}$, the maximum nonnegative feasible step along $\underline{p}_k$.
  Determine a positve step length α_k, for $f(\underline{x}^k + \alpha_k \underline{p}_k) < f(\underline{x}^k)$ and $\alpha_k \leq \hat{\alpha}$. If $\alpha_k < \hat{\alpha}$, go to step 7.
- **Step 5:** [Add a constraint to the working set]: If α_k is a step to a constraint with index r, add r to I^k, and updating the assiciated quantities accordingly. Go to step 7.

- **Step 6:** [Delete a constraint from the working set]: Choose a constraint with index s such that s corresponds to $min\ \lambda_i,\ i = 1, \ldots,\ t0_k$. Delete s from I^k, update the associated quantities, set $k = k + 1$, and go to step 1.

- **Step 7:** [Update the estimate of the solution]: Set $\underline{x}^{k+1} = \underline{x}^k + \alpha_k \underline{p}_k$, set $k = k+1$, and go to step 1.

The convergence test of the algorithm is satisfies when the feasible direction $\underline{p}_k = \underline{0}$ and all the Lagrange multiplers $\lambda_i,\ i = 1, \ldots,\ t0_k$ are nonnegatives.

2.5.2 Computation of the search direction and step length

In an active set method, the search direction is constructed to lie in a subspace defined by the working set. Thus, the search direction is given by $\underline{p}_k = Z_k \underline{p}_z$. Let i be the index of the constraint that is not in the working set at $\underline{x}^k$, so that $i \notin I^k$. If $\underline{a}_i^T \underline{p}_k \geq 0$, any positive move along $\underline{p}_k$ will not violate the constraint. So if $\underline{a}_i^T \underline{p}_k$ is nonnegative for all such constraints, the constraints that are not in the working set imopse no restriction of the step length. On the other hand, if $\underline{a}_i^T \underline{p}_k < 0$, there is a critical step γ_i where consraints become "active". The value of γ_i is given by

$$\gamma_i = \frac{\hat{b}_i - \underline{a}_i^T \underline{x}^k}{\underline{a}_i^T \underline{p}_k}, \quad i \notin I^k \text{ and } \underline{a}_i^T \underline{p}_k < 0$$

Let

$$\hat{\alpha} = \begin{cases} \min(\gamma_i), & \text{if } \underline{a}_i^T \underline{p}_k < 0,\ i \notin I^k \\ +\infty, & \text{if } \underline{a}_i^T \underline{p}_k \geqslant 0,\ i \notin I^k \end{cases}$$

the value of $\hat{\alpha}$ is the maximum nonnegative feasible step that can be taken along $\underline{p}_k$, and it is taken as an upper bound on the feasible step length α_k. When $\alpha_k < \hat{\alpha}$, the working set at the next iteration is not altered. When $\alpha_k = \hat{\alpha}$, the working set must be modified to reflect the new constaint that is added to the working set.

2.5.3 Change in the working set

When a constraint is added to the working set, a new row is added to $\hat{A}_k^T$. When a constraint is deleted from the working set, one of the rows of $\hat{A}_k^T$ is removed. In either case, it would clearly be inefficient to recompute Z_k from scratch. Rather, it is possible to modify the representation of Z_k to correspond to the new working set. We discuss how to update Z_k and other matrices following a single change in the working set. We limit our discussion to the case in which the step is computed with the null space method.
Suppose that $\hat{A}_k^T$ has $t0$ linearly independent rows and assume that the basis Y_k and Z_k are defined by means of a QR factorization of $\hat{A}_k$. Thus,

$$\hat{A}_k\pi = Q \begin{bmatrix} R \\ O \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ O \end{bmatrix} = Q_1 R, \tag{2.27}$$

where π is a permutation matrix, R is square upper triangular nonsingular matrix, $Q = [Q_1, Q_2]$ is $n \times n$ orthogonal, and Q_1, Q_2 are $n \times \dot{m}$ and $n \times (n - \dot{m})$, respectively. We choose Z_k to be simply $Z_k = Q_2$. Suppose that one constraint is added to the working set at the next iteration, so that the new constraint matrix $\overline{A}_k = \begin{bmatrix} \hat{A}_k & \underline{a} \end{bmatrix}$, where $\underline{a}$ is column vector of length n such that $\overline{A}_k$ remains a full column rank. We now descibe an economical way to update Q and R factors in (2.27) to obtain the new factors for the expanded matrix $\overline{A}_k$.
Note first that, since $Q_1 Q_1^T + Q_2 Q_2^T = I$, we have

$$\overline{A} \begin{bmatrix} \pi & \underline{0} \\ O & 1 \end{bmatrix} = \begin{bmatrix} \hat{A}_k\pi & \underline{a} \end{bmatrix} = Q \begin{bmatrix} R & Q_1^T\underline{a} \\ O & Q_2^T\underline{a} \end{bmatrix}. \tag{2.28}$$

We can now define the orthgonal matrix $\check{Q}$ that transform the vector $Q_2^T\underline{a}$ to a vector in which all elements except the first are zero. That is, we have

$$\check{Q}(Q_2^T\underline{a}) = \begin{bmatrix} \nu \\ \underline{0} \end{bmatrix}, \tag{2.29}$$

where ν is scalar. Since $\check{Q}$ is orthogonal, we have $\| Q_2^T \underline{a} \| = | \nu |$. By substituting (2.29) in (2.28), we have

$$\overline{A} \begin{bmatrix} \pi & \underline{0} \\ O & 1 \end{bmatrix} = Q \begin{bmatrix} R & Q_1^T \underline{a} \\ O & \check{Q}^T \begin{bmatrix} \nu \\ \underline{0} \end{bmatrix} \end{bmatrix} = Q \begin{bmatrix} I & \underline{0} \\ O & \check{Q}^T \end{bmatrix} \begin{bmatrix} R & Q_1^T \underline{a} \\ O & \nu \\ O & \underline{0} \end{bmatrix}. \tag{2.30}$$

This factorization has the form $\overline{A}\overline{\pi} = \overline{Q} \begin{bmatrix} \overline{R} \\ O \end{bmatrix}$, where

$$\overline{\pi} = \begin{bmatrix} \pi & \underline{0} \\ O & 1 \end{bmatrix}, \overline{Q} = Q \begin{bmatrix} I & \underline{0} \\ O & \check{Q}^T \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \check{Q}^T \end{bmatrix}, \overline{R} = \begin{bmatrix} R & Q_1^T \underline{a} \\ O & \nu \end{bmatrix}.$$

We can choose $\overline{Z}_k$ to be the last n-$\dot{m}$-1 columns of $Q_2 \check{Q}^T$. If we know Z_k explicitly and need the explicit representation of $\overline{Z}$, we need to account for the cost of obtaining $\check{Q}$ and the cost of forming the product $Q_2 \check{Q}^T = Z_k \check{Q}^T$. Because of the special structure of $\check{Q}$, this cost is of order $n(n - \dot{m})$, compared to the cost of computing (2.27) which is of order $n^2 \dot{m}$. The updating strategy is less expensive, especially when the null space is small $(n - \dot{m} << n)$.

An uppdating technique can also be designed for the case in which a row is deleted from $\hat{A}^T$. For detials of this process see (Nocedal and Wright, 2006).

2.6 Line search and trust-region methods

Line search methods and trust-region methods are important methods for solving optimization problems. They both generate steps with the help of a quadratic model of the objective function, but they use this model in different ways. In this section we descibe these methods briefly for more details, see (Nocedal and Wright, 2006)

2.6.1 Line search Methods

The line search methods are the oldest and most widely used techniques for solving optimization problems specially in unconsrained optimization. At each iteration k, $k =$

$1, 2, \ldots,$ the strategy in the line search methods is to choose a search direction $\underline{p}_k$ and search along this direction from a current iterate $\underline{x}_k$ for a new iterate with a lower function value. The distance α_k to move along $\underline{p}_k$ can be found by approximately solving the following one dimensional minimization problem:

$$\min_{\alpha>0} f(\underline{x}_k + \alpha \underline{p}_k). \tag{2.31}$$

By solving this problem exactly, we would derive the maximum benefit from the direction $\underline{p}_k$, but an exact solution may be expensive and usually unnecessary. Instead, the line search algorithm generates a limited number of trial step lengths until it finds one that loosely approximates the minimum of equation (2.31). The next iteration is defined by

$$\underline{x}_{k+1} = \underline{x}_k + \alpha_k \underline{p}_k. \tag{2.32}$$

The success of line search methods depends on effective choice of the direction and, the step length. Most line search directions require $\underline{p}_k$ to be a descent direction, i.e, one for which $\underline{p}_k^T \nabla f(\underline{x}_k) < 0$, which guarantees that the function $f(\underline{x})$ can be reduced along this direction. A simple condition we can impose on α_k is that $f(\underline{x}_k + \alpha_k \underline{p}_k) < f(\underline{x}_k)$. This requirement is not enough to produce convergence to $\underline{x}^\star$, $\underline{p}_k$ must satisfy the stronger Wolfe conditions

$$f(\underline{x}_k + \alpha_k \underline{p}_k) \leq f(\underline{x}_k) + c_1 \alpha_k \nabla f(\underline{x}_k)^T \underline{p}_k \tag{2.33}$$

$$|\nabla f(\underline{x}_k + \alpha_k \underline{p}_k)^T \underline{p}_k| \leq c_2 \mid \nabla f(\underline{x}_k)^T \underline{p}_k \mid, \tag{2.34}$$

where $c_1,\ c_2$ are constants, such that $0 < c_1 < c_2 < 1$.

2.6.2 trust-region methods

The concept of trust-region first appeared in the papers of (Levenberg, 1944) and (Maquardt, 1963) for solving nonlinear least squars problems. Trust-region methods are iterative, in each iteration, an approximation of the objective function $f(\underline{x}_k)$ by a model $Q_k(\underline{p}_k)$ is computed in a neighbourhood of the current iterate $\underline{x}_k$. This neighbourhood is called the trust-region. The model Q_k should be constructed so that it is easier to handle

than $f(\underline{x}_k)$.

Assume that $f(\underline{x}_k)$ is a C^2 function. Using Tayler's expansion of $f(\underline{x})$ at $\underline{x}_k$, let

$$Q_k(\underline{p}_k) = f_k + g_k^T \underline{p}_k + \frac{1}{2}\underline{p}_k^T \nabla^2 f(\underline{x}_k + t\underline{p}_k)\underline{p}_k, \tag{2.35}$$

where $f_k = f(\underline{x}_k)$, $g_k = \nabla f(\underline{x}_k)$ and t is scalar in the interval $(0, 1)$. By using an approximation B_k to the Hessiam matrix, then Q_k is defined as

$$Q_k(\underline{p}_k) = f_k + g_k^T \underline{p}_k + \frac{1}{2}\underline{p}_k^T B_k \underline{p}_k. \tag{2.36}$$

The difference between $Q_k(\underline{p}_k)$ and $f(\underline{x}_k + \underline{p}_k)$ is $O(\| \underline{p}_k \|^2)$ which is small when $\underline{p}_k$ is small. To obtain the step $\underline{p}_k$, we seek a solution of the subproblem

$$\min_{\underline{p}_k \in \mathbb{R}^n} Q_k(\underline{p}_k) = f_k + g_k^T \underline{p}_k + \frac{1}{2}\underline{p}_k^T B_k \underline{p}_k, \quad \text{subject to } \| \underline{p}_k \| \leq \Delta_k, \tag{2.37}$$

where Δ_k is the trust-region radius. When B_k is positive definite and $\| B_k^{-1} g_k \| \leq \Delta_k$, the solution of (2.37) is simply the unconsrained minimum $\underline{p}_k^\beta = -B_k^{-1} g_k$ of the quadratic $Q_k(\underline{p}_k)$. In this case, $\underline{p}_k^\beta$ is called full step. The solution of (2.37) is not so obvious in the other cases, but it can usually be found without too much computational expence, in any case, we need only an approximation solution to obtain the convergence.

One of the key ingredients in a trust-region algorithm is the strategy for choosing the trust-region radius Δ_k at each iteration. We base this choice on the agreement between the model Q_k, and the objective function f at previous iterations. Given the step $\underline{p}_k$, we define the ratio

$$\text{RATIO} = \frac{f(\underline{x}_k) - f(\underline{x}_k + \underline{p}_k)}{Q_k(0) - Q_k(\underline{p}_k)}. \tag{2.38}$$

The numerator is called the actual reduction and the denominator is called the predicted reduction. We observe that the predicted reduction will always be nonnegative. Hence, if RATIO is negative, the new objective value $f(\underline{x}_k + \underline{p}_k)$ is greater than the current value $f(\underline{x}_k)$, so the step must be rejected. On the other hand, if RATIO is close to one, there is a good agreement between the model Q_k and the objective function f over this step, so it is safe to expand the trust-region radius for the next iteration. If RATIO is positive but significantly smaller than one, we do not alter the trust-region radius, but if it is close to zero or negative, we shrink the trust-region radius Δ_k at the next iteration. The following

algorithm (Nocedal and Wright, 2006) describes the process :

Algorithm 2.2 [trust-region algorithm]
Given $\hat{\Delta} > 0$, $\Delta_0 \in (0, \hat{\Delta})$ and $\eta \in [0, \frac{1}{4}]$
For $k = 0, 1, \ldots,$ until convergence
Obtain $\underline{p}_k$ by solving (2.37)
Evaluate RATIO from (2.38)
If RATIO $< \frac{1}{4}$
$\Delta_{k+1} = \frac{1}{4}\Delta_k$
Else
if RATIO $> \frac{3}{4}$ and $\| \underline{p}_k \| = \Delta_k$
$\Delta_{k+1} = \min(2\Delta_k, \hat{\Delta})$
else
$\Delta_{k+1} = \Delta_k$
end(if)
End(if)
If RATIO $> \eta$
$\underline{x}_{k+1} = \underline{x}_k + \underline{p}_k$
Else
$\underline{x}_{k+1} = \underline{x}_k$
End(if)
End(for).

Here $\hat{\Delta}$ is an overall bound on the step length. Note that the radius is increased only if $\| \underline{p}_k \|$ actaully reaches the boundary of the trust-region. To turn algorithm (2.2) into a practical algorithm we need to focus on solving the trust-region subproblem(2.37). There are many methods used to solve the subproblem(2.37), such as the method of (More and Sorensen, 1983), Dogleg method see (Conn et al., 2000), truncated conjugate gradient method see (Conn et al., 2000). In this thesis, we use the truncated conjugate gradient method.

2.7 The truncated conjugate gradient method

In this section we describe a technique that is use to solve the subproblem (2.37), namely the conjugate gradient method. The method is used to solve the trust-region subproblem discussed in section 4.4.

The conjugate gradient method produces a piecewise linear path in $\mathbb{R}^n$, starting at $\underline{x}_k = \underline{x}_k + \underline{p}_0$, where $\underline{p}_0 = 0$. For $j \geq 1$, let $\underline{x}_k + \underline{p}_j$ be a point in the path at the end of the j-th line segment. It has the form

$$\underline{x}_k + \underline{p}_j = \underline{x}_k + \underline{p}_{j-1} + \alpha_j \underline{s}_j \ , \ j \geq 1, \tag{2.39}$$

where $\underline{s}_j$ is the direction of the line segment, and α_j is the step length. The path is truncated at $\underline{x}_k + \underline{p}_{j-1}$ if $\| \nabla Q_k(\underline{x}_k + \underline{p}_{j-1}) \|$ is sufficiently small, or if $\| \underline{p}_{j-1} \| = \Delta_k$. When the j-th line segment of the path is constructed, its direction is defined by

$$\underline{s}_j = \begin{cases} -\nabla Q_k(\underline{x}_k) \ , & j = 1 \\ -\nabla Q_k(\underline{x}_k + \underline{p}_{j-1}) + \beta_{j-1}\underline{s}_{j-1}, & j \geq 2, \end{cases} \tag{2.40}$$

where β_{j-1} is the ratio $\| \nabla Q_k(\underline{x}_k + \underline{p}_{j-1}) \|^2 \ / \ \| \nabla Q_k(\underline{x}_k + \underline{p}_{j-2}) \|^2$. The step length α_j is chosen to minimize $Q_k(\underline{x}_k + \underline{p}_j)$ subject to $\| \underline{p}_j \| \leq \Delta_k$ for each j. The following algorithm (Conn et al., 2000), descibes the method .

Algorithm 2.3 [The Steihaug-Toint truncated conjucate gradient algorithm]

Let $\underline{p}_0 = 0$, $\underline{g}_0 = \underline{g}_k$, $\underline{s}_0 = -\underline{g}_0$, for $j = 0, 1, \ldots,$

Repeat until convergence

set $\kappa_j = \underline{s}_j^T B_k \underline{s}_j$

if $\kappa_j \leq 0$

compute α_j the positive root of $\| \underline{p}_j + \alpha_j \underline{s}_j \| = \Delta_k$

set $\underline{p}_{j+1} = \underline{p}_j + \alpha_j \underline{s}_j$, stop

else

set $\alpha_j = \underline{g}_j^T \underline{s}_j / \kappa_j$

if $\| \underline{p}_j + \alpha_j \underline{s}_j \| \geq \Delta_k$

compute α_j the positive root of $\| \underline{p}_j + \alpha_j \underline{s}_j \| = \Delta_k$

set $\underline{p}_{j+1} = \underline{p}_j + \alpha_j \underline{s}_j$, stop

end(if)

set $\underline{p}_{j+1} = \underline{p}_j + \alpha_j \underline{s}_j$

$\underline{g}_{j+1} = \underline{g}_j + \alpha_j B_k \underline{s}_j$

$\beta_j = \| \nabla Q_k(\underline{x}_k + \underline{p}_{j+1}) \|^2 \; / \; \| \nabla Q_k(\underline{x}_k + \underline{p}_j) \|^2$

$\underline{s}_{j+1} = -\underline{g}_{j+1} + \beta_j \underline{s}_j$

end(if)

End(repeat).

2.8 Interpolation model-based derivative-free methods

Derivative-free methods can be classified into two classes. The direct search methods and model-based methods. The direct search methods explore the domain using symmetric rules. Following (Lewis et al., 2000), direct search methods are classified into three categories.

- Pattern search methods: A Pattern search method chooses a certain set of search directions $\underline{k}_j^j,\ j = 1, 2, \ldots,$ at each iteration and evaluates the objective function at a given step length along each of these directions. The resulting coordinate points form a "frame" around the current iterate. If a point with a significantly lower function value is found, it is adopted as the new iterate and the center of the frame is shifted to this new point.

- Simplex search direction methods: These methods take the name from the fact that at any stage of the algorithm, we keep track of $n+1$ points of interest in $\mathbb{R}^n$, whose convex hull form a simplex. In a single iteration of the algorithm, we seek to remove the vertex the with worst function value and replace it with a better value. The new point is obtained by reflecting, expanding or contracting the simplex along the line

jointing the worst vertex with the centre of the remaining vertices. An example of simplex search direction methods is the Nelder-Mead method (Conn et al., 2009).

- Implicit Filtering method (Conn et al., 2009): The implicit Filtering approach in its simplest form, is a variant of the steepest descent algorithm with line search, in which the gradient is replaced by a finite difference estimate.

The second class is the model-based methods. The model-based methods construct linear or quadratic models for the objective function and define the next iterate by seeking to minimize the model inside a trust-region. Most algorithms proposed in the literature are based on the quadratic models, such as DFO (Derivative-Free Optimization, (Conn et al., 1998)), CONDOR (a constrained derivative-free parallel optimizer for continuous high computing load, noisey objective function, Frank and Bersini (2005)), UOBYQA (Unconstrained Optimization BY Quadratic Approximation, (Powell, 2002)), NEWUOA (NEW Unconstrained Optimization BY Qudratic Approximation (Powell, 2006)), and BOBYQA (Bound Optimization BY Quadratic Approximation (Powell, 2009)).

2.8.1 Review of History of Model-based Derivative-Free Optimization Methods

Although it is not known exactly when the idea of derivative free optimization methods was first introduced, the approach of using direct search-methods arise in the 1950, see (Conn et al., 1997a). A detailed review of the historical development of derivative free optimization methods can be found in (Conn et al., 1997a). The idea of employing available objective function values $f(\underline{x})$ for building a quadratic model by interpolation was firstly proposed by Winfield (Winfield, 1969, 1973). This model is assumed to be valid in a neihbourhood of the current iterate, which is described as a trust-region whose raduis iteratively adjusted. The model is then minimized within a trust-region, hopefully yielding a point with a lower function value. As the algorithm proceeds and more objective function values become available, the set of the points defining the interpolation model is updated in such a way that always contains the points closest to the current iterate.

Winfield recognizes the difficulty that the interpolation points must have certain geometric properties, also he does not seem to consider what is happents if these properties are not satisfied.

In 1994, Powell proposed a method (COBYLA) for constrained optimization, whose idea is close to that of Winfield. In this proposal, the objective function and the constraints are approximated by linear multivariate models. He also explores Winfield's idea further by describing an algorithm for unconstrained optomization without derivative using quadratic multivariate interpolation model of the objective function in a trust-region framework, UOBYQA, (Powell, 2002). A variant of this quadratic model using Newton fundemental polynomials was discussed by (Conn et al., 1998). In 2001 and 2005, (Faul, 2001) (Oeuvray, 2005) implemented two different unconstrained derivative free algorithms based on radial basis functions. Also, in 2005, (Frank and Bersini, 2005) (CONDOR) proposed a variant of Powell's (Powell, 2002) algorithm for unconstrained and easy constrained cases. (Powell, 2006, 2009), proposed algorithms for unconstrained and simple bound constrained optimization without derivative based on the least Frobenius norm. The first convergence theory for the model-based derivative free optimization were presented by (Conn et al., 1997a). They also described some alternative techniques to strengthen the geometric properties of the set of the interpolation points. Sheinberg, and Vicente also described techniques about interpolation error estimates and about the geometry of interpolation points (Sheinberg and Vicente, 2006, 2007, 2008).

2.8.2 Multivariate Interpolation

Interpolation, is a technique for constructing a polynomial which goes through a given set of data points. The simplest kind of interpolation is the interpolation by means of univariate polynomials. Different formulae for polynomial interpolation have been given, as for example, Newton and Lagrange methods. In this section we consider the problem of interpolating a given function $f(\underline{x})$, $\underline{x} \in \mathbb{R}^n$ by a polynomial (quadratic) at a chosen set of interpolation points $Y = \left\{\underline{y}_i\right\}_{i=1}^{s} \subset \mathbb{R}^n$. In other words, we need to find a quadraric

polynomial $Q(\underline{x})$, for which

$$Q(\underline{y}_i) = f(\underline{y}_i),\ i = 1, 2, \ldots, s. \tag{2.41}$$

We say that a set of points Y can be interpolated by a polynomial of a certain degree, if for any function f there exists a polynomial of degree r such that equations (2.41) hold for all points in the set. In the case of univariate interpolation, any set of distinct points can be interpolated by a polynomial of appropriate degree. For example, any three distinct points can be interpolated by a quadratic polynomial. In multivariate interpolation, however, this is not the case. For example, to obtain a unique quadratic interpolation of a function in two variables one needs six interpolation points, but a set of six interpolation points on a line can not interpolate a quadratic polynomial.

Definition 3.1 A set of points Y is called poised with respect to a given subspace of polynomials, if it can be interpolated by a polynomial from this space.

For example, suppose $n = 2$, and Y is a set of six points on a unit circle. Then Y cannot be interpolated by a polynomial of the form: $a_0+a_1x_1+a_2x_2+a_{11}x_1^2+a_{12}x_1x_2+a_{22}x_2^2$. Thus, Y is not poised with respect to the space of quadratic polynomials. On the other hand, Y can be interpolated by a polynomial of the form:$a_0+a_1x_1+a_2x_2+a_{11}x_1^2+a_{12}x_1x_2+a_{22}x_1^3$, thus, Y is poised in an appropriate subspace of the space of cubic polynomials.

Definition 2.2 A set of points Y is called well-poised, if it remains poised under small perturbations. For example, if $n = 2$, six points almost on a line may may be poised. However, since some small perturbation of the points might make them aligned, so Y is not a well-poised set, the interolation in this case will be very ill-conditioned and is likely to provide a very bad approximation for the function.

Suppose $\{\phi_i(.)\}_{i=1}^s$ is a basis in the space of quadratic polynomials, then any quadratic polynomial $Q(\underline{x})$ can be written as $Q(\underline{x}) = \sum_{i=1}^s \alpha_i\phi_i(\underline{x})$, for some $\alpha = (\alpha_1, ..., \alpha_s)$. The interpolation conditions can be written as the system of linear equations in α

$$\sum_{i=1}^{s} \alpha_i\phi_i(y_j) = f(y_j),\ j = 1, 2, \ldots, s. \tag{2.42}$$

The coefficient matrix for this system is:

$$\Phi(Y) = \begin{pmatrix} \phi_1(y_1) & \cdots & \phi_1(y_s) \\ \vdots & \ddots & \vdots \\ \phi_s(y_1) & \cdots & \phi_s(y_s) \end{pmatrix}. \tag{2.43}$$

For a given set of points and a set of function values, an interpolatin polynomial exists and is unique if and only if $\Phi(Y)$ is square and non-singular. The cardinality of Y and that of the basis $\{\phi_i(.)\}$ are the same and equal to s. For full quadratic interpolation $s = \frac{1}{2}(n+1)(n+2)$. Note that the nonsigularity of $\Phi(Y)$ is independent of the choice of the polynomial basis, as long as the space that is spanned by the basis is fixed. With respect to a given space of polynomials, a set of points Y is poised if $\Phi(Y)$ is non-singular. If Y is poised, then, theoretically, we can solve the linear system $\Phi(Y)$ and find the interpolation polynomial. However, numerically the the matrix $\Phi(Y)$ is often ill-conditioned even when it is non-singular. Conditioning of $\Phi(Y)$, clearly depends on the choice of the basis $\{\phi_i(.)\}$. For example, one can choose a basis such that $\Phi(Y)$ is an identity matrix. Such basis is called Lagrange fundamental polynomial basis.

2.8.3 Lagrange fundamental polynomials

Lagrange fundamental polynomials play an important role in the theory of univariate and multivariate polynomial interpolation. They are also useful in the model-based derivative-free optimization, because the interpolation model is easily expressed through the basis of the Lagrange polynomials.

Lagrange gave the following interpolation polynomial

$$Q(\underline{x}) = \sum_{i=1}^{s} f(y_i)\ell_i(\underline{x}), \tag{2.44}$$

for interpolating $f(\underline{x})$ at $\underline{y}_i,\ i = 1, 2, \ldots,$s, where $\ell_i's$ are the Lagrange basis polynomials that satisfy $\ell_i(\underline{x}_i) = 1,\ \ell_i(\underline{x}_j) = 0,\ i \neq j$. We will construct our polynomial basis $\ell_i(\underline{x}),\ i = 1, 2, \ldots, s$, iteratively. Assuming that we already have a polynomial

$$Q(\underline{x}) = \sum_{i=1}^{k} f(y_i)\ell_i(\underline{x}), \tag{2.45}$$

interpolating k points, we will add to it a new polynomial ℓ_{k+1} which does not destroy what we have already done. That is, the value of ℓ_{k+1} must be zero for $\underline{x} = \underline{x}_1, \underline{x}_2, \ldots, \underline{x}_k$ and $\ell_{k+1}(\underline{x}_{k+1}) = 1$. This is easily done in the univariate case:

$$\ell_{k+1}(\underline{x}) = \frac{(\underline{x} - \underline{x}_1)(\underline{x} - \underline{x}_2)\ldots(\underline{x} - \underline{x}_k)}{(\underline{x}_{k+1} - \underline{x}_1)(\underline{x}_{k+1} - \underline{x}_2)\ldots(\underline{x}_{k+1} - \underline{x}_k)}. \tag{2.46}$$

But, in the multivariate interpolation case, it becomes difficult. In the multivariate case we must find a new polynomial ℓ_{k+1} which is perpendicular to $\ell_i(\underline{x}),\ i = 1, 2, \ldots, k$, with respect to the points $\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_k$. Any multiple of ℓ_{k+1} added to the previous ℓ_i must leave the value of $\ell_i,\ i = 1, 2, \ldots, k$, unchanged at the k points $\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_k$. We can consider the polymials $\ell_i,\ i = 1, 2, \ldots, k$ as "vectors", and construct a new vector ℓ_{k+1} which is perpendicular to all $\ell_i,\ i = 1, 2, \ldots, k$, using a version of Gram-Schmidt orthognalization procedure adapted for polynomials.

The Gram-Schmidt orthognalization procedure takes a set of independent polynomials $\ell_{1old}, \ell_{2old}, \ldots, \ell_{sold}$, and converts them into a set of orthogonal vectors $\ell_1, \ell_2, \ldots, \ell_s$, by the following algorithm (Frank and Bersini, 2005).

Alorithm 2.4 [The Gram-Schmidt orthognalization Algorithm]

step 1: Initialization: set $k = 1$

step 2: Normalization:

$$\ell_k(\underline{x}) = \ell_{kold}(\underline{x})/\ell_{kold}(\underline{x}_k), \quad | \ \ell_{kold}(\underline{x}_k) \ | \neq 0, \tag{2.47}$$

step 3: Orthogonalization:

for $j = 1$ to s, $j \neq k$ do

$$\ell_{j_{old}}(\underline{x}) = \ell_{j_{old}}(\underline{x}) - \ell_{j_{old}}(\underline{x}_k)\ell_k(\underline{x}) \tag{2.48}$$

end(for)

step 4: Loop increment:

set $k = k + 1$, if $k < s$ go step 2.

After the complection of the algorithm, we discard all the $\ell_{j_{old}}$ and replace them by ℓ_j^s for the next iteration. The set of polynomials $\{\ell_1, \ell_2, \ldots, \ell_m\}$ are simply initialized

with monomials of a polynomial of dimension n. For example, if $n = 2$, we obtain $\ell_1(\underline{x}) = 1,\ \ell_2(\underline{x}) = x_1,\ \ell_3(\underline{x}) = x_2,\ \ell_4(\underline{x}) = x_1^2,\ \ell_5(\underline{x}) = x_1x_2,\ \ell_6(\underline{x}) = x_2^2, \ldots$. In equation (2.47) there is a division, to improve the stability of the algorithm, we must do pivoting, that selects a salubrious pivot element for the division in (2.47) so that the denominator of (2.47) is far from zero. After comletion of the algorithm, we have $\ell_i(\underline{x}_j) = \delta_{ij},\ i,\ j = 1, \ldots,\ s$.

2.8.4 Interpolation model-based derivative-free algorithm

Various algorithms have recently been devoloped for derivative-free optimiztion COBYLQ, (Powell, 1994), UOBYQA, (Powell, 2002), NEWUOA, (Powell, 2006), BOBYQA, (Powell, 2009), CONDOR, (Frank and Bersini, 2005), DFO (Conn et al., 1998). Most of the proposed algorithms in the literature are based on quadratic models. Model-based derivative-free optimization algorithms are extension to trust-region algorithms. We will discuss in detial model-based derivative-free optimization algorithm for the unconstrianed case. Let $\underline{x}_{opt}$ be a point such that $f(\underline{x}_{opt})$ is least value of f so far. Condsider the quadratic model

$$Q_k(\underline{x}_{opt} + \underline{p}_k) = c + \underline{p}_k^T \underline{g} + \frac{1}{2}\underline{p}_k^T G \underline{p}_k. \tag{2.49}$$

We can not define $\underline{g} = \nabla f(\underline{x}_{opt})$ and $G = \nabla^2 f(\underline{x}_{opt})$, because the derivatives are not available. Instead, we determine the scalar c, the vector $\underline{g}$ and the symmetric matrix $G \in \mathbb{R}^{n\times n}$ by imposing the interpolation conditions

$$Q_k(\underline{x}_i) = f(\underline{x}_i),\ i = 1, 2, \ldots, s, \tag{2.50}$$

where $\underline{x}_i \in Y = \{\underline{x}_1, \ldots, \underline{x}_s\}$. Thus, $s = 1+n+\frac{1}{2}n(n+1) = \frac{1}{2}(n+1)(n+2)$. In this case equation (2.50) can be written as a square linear system of equations in the coefficients of the model. If we choose the interpolation points $\underline{x}_1,\ \underline{x}_2, \ldots,\ \underline{x}_s$, so that the linear system is nonsingular (Y is poised), then the model Q_k will be determined uniquely. Once Q_k is formed, we compute the step $\underline{p}_k$ by approximately solving the trust-region subproblem (2.49). If $\underline{x}_{opt} + \underline{p}_k$ gives sufficient reduction in the value of the objective function, then the new iterate is defined as $\underline{x}_{opt} = \underline{x}_{opt} + \underline{p}_k$, the trust-region Δ is updated, and the new

iteration commences. Otherwise, the step is rejected, and the interpolation set Y may be improved or the trust-region shrunk. To reduce the cost of the algorithm, we update the model Q_k at every iteration, rather than recomputing it from scratch. In practice, we choose a convenient basis for the space of quadratic polynomials, the most common choice being Lagrange and Newton interpolation polynomials. In this thesis we use the Lagrange polynomials. The properties of these bases can be used both to measure appropriateness of the sample set Y and to change this set if necessary. A complete algorithm that treats all these issues effectively is more complicated than the trust-region algorithm. An outline of such algorithm is given below see (Nocedal and Wright, 2006).

Algorithm 2.5 [model-based derivative-free algorithm]

step 1: Choose a poised interpolation set $Y = \{\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_s\}$, select a point $\underline{x}_{opt} \in Y$ such that $f(\underline{x}_{opt}) \leq f(\underline{x}_i),\ i = 1, 2, \ldots, s.$

Choose the initial trust-region Δ_1, a constant $\eta \in (0, 1)$, set $k = 1$.

step 2: Repeat until convergence test satisfied:

Compute the step $\underline{p}_k$ by minimizing equation (2.49), define the trial step $\underline{x}^+_{opt} = \underline{x}_{opt} + \underline{p}_k$.

Compute the RATIO from (2.38).

step 3: If RATIO $\geq \eta$

replace an element of Y by $\underline{x}^+_{opt}$

choose $\Delta_{k+1} \geq \Delta_k$

set $\underline{x}_{opt} = \underline{x}^+_{opt}$

set k=k+1, go to step 2.

else

if the set Y need not be improved

choose $\Delta_{k+1} < \Delta_k$, set $k = k + 1$, go to step 2

end(if)

end(if)

setp 4: Invoke a geometry improvement procedure to update Y.

One point of Y is replaced by some other point with the goal of improving the condition of the system (2.50), let $\underline{y}$ be the new point, set $\Delta_{k+1} = \Delta_k$

if $f(\underline{y}) < f(\underline{x}_{opt})$

set $\underline{x}_{opt} = \underline{y}$

end(if)

set $k = k + 1$, go to step 2

end(repeat).

To make this outline algorithm a practical algorithm, we must discuss how to improve the geometry of the set Y. Also, we must solve the subproblem (2.49), and we must also dermine the test of convergence. In chapter 4 we will discuss all these issues in detail.

Chapter 3

Least Frobenius Norm Method for Updating Quadratic Models

3.1 Introduction

The quadratic model of the objective function is highly useful for obtaining a fast rate of convergence in the derivative-free optimization algorithms, because it contains the curvature of the objective function. On the other hand, the use of full quadratic model limits the size of the problem that can be solved, because each full quadratic model has $\frac{1}{2}(n+1)(n+2)$ independent parameters. One idea that overcomes this difficulty was proposed by Powell (Powell, 2004a, 2006). Powell constructs a quadratic model from a few data, and uses the remaining freedom in the model to minimize the Frobenius norm of the second derivative matrix of the change to the model. This variational problem is expressed as the solution of a $(m+n+1)\times(m+n+1)$ system of linear equations, where m is the number of interpolation points which satisfy $n+2 \leq m \leq \frac{1}{2}(n+1)(n+2)$.

In this chapter we discuss in detial the idea described in the above paragraph, it is the main idea of Powell's algorithms (NEWUOA, BOBYQA) and thus of LCOBYQA algorithm, since LCOBYQA is an extension to Powell algorithms for linearly inequality constrained problems. LCOBYQA will be presented in Chapter 4.

The rest of this chapter is oragnized as follows. The existence and uniqueness of the variational problem is discussed in section 3.2. Lagrange fundamental polynomials, which play a vital role in preserving the nonsingularity of the system of the variational problem, we address this topic in section 3.3. In section 3.4, we discuss the updating of the matrix of the variational linear system. The algorithm which applies the idea of least Frobenius norm update for unconstrained problems, namelly NEWUOA (Powell, 2006), is preseted in section 3.5. Finally, in section 3.6, we explore BOBYQA (Powell, 2009) algorithm which is an extension of NEWUOA to simple bound constraints.

3.2 The solution of the variational problem

Let $\underline{x}_i,\ i = 1, 2, \ldots, m$ be the old interpolation points.

Let the quadratic model

$$Q_{old}(\underline{x}) = c_{old} + (\underline{x} - \underline{x}_0)^T \underline{g}_{old} + \frac{1}{2}(\underline{x} - \underline{x}_0)^T G_{old}(\underline{x} - \underline{x}_0),\ \underline{x} \in \mathbb{R}^n, \tag{3.1}$$

be given, and satisfies the interpolation conditions

$$Q_{old}(\underline{x}_i) = f(\underline{x}_i),\ i = 1, 2, \ldots, m,$$

where $\underline{x}_0$ is a fixed point.

In this section we consider the change that is made to $Q_{old}(\underline{x})$ in each iteration that alters the set of interpolation points. Let the new interpolation points be $\underline{x}_t^+,\ \underline{x}_i^+ = \underline{x}_i,\ i \in \{1, 2, \ldots, m\} \setminus \{t\}$, where t is the index of the point that will be removed.

Let

$$Q_{new}(\underline{x}) = c^+ + (\underline{x} - \underline{x}_0)^T \underline{g}^+ + \frac{1}{2}(\underline{x} - \underline{x}_0)^T G^+(\underline{x} - \underline{x}_0),\ \underline{x} \in \mathbb{R}^n, \tag{3.2}$$

be the new quadratic model that satisfies the conditions

$$Q_{new}(\underline{x}_i^+) = f(\underline{x}_i^+),\ i = 1, 2, \ldots, m. \tag{3.3}$$

Therefore, after satisfying the conditions (3.3), we employ the remaining freedom in Q_{new} to minimize the Frobenius norm of the second derivative matrix of the change of the

model, i.e, we employ the remaining freedom in Q_{new} to minimize the Frobenius norm of $\nabla^2 D(\underline{x})$, where

$$D(\underline{x}) = Q_{new}(\underline{x}) - Q_{old}(\underline{x}) = c + (\underline{x} - \underline{x}_0)^T \underline{g} + \frac{1}{2}(\underline{x} - \underline{x}_0)^T G(\underline{x} - \underline{x}_0),\ \underline{x} \in \mathbb{R}^n, \quad (3.4)$$

subject to the coditions (3.3). This problem can be written as:

$$\min\ \frac{1}{4} \parallel G^+ - G_{old} \parallel_F^2 = \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} G_{ij}^2, \quad (3.5)$$

subject to

$$c + (\underline{x}_i^+ - \underline{x}_0)^T \underline{g} + \frac{1}{2}(\underline{x}_i^+ - \underline{x}_0)^T G(\underline{x}_i^+ - \underline{x}_0) = \left[f(\underline{x}_i^+) - Q_{old}(\underline{x}_i^+) \right] \delta_{it}, \quad i = 1, 2, \ldots, m. \quad (3.6)$$

This problem is a convex qradratic problem. Therefore, from the KKT conditions, there exist unique Lagrange multipliers $\lambda_k,\ k = 1, 2, \ldots, m$, such that the first derivative of the expression

$$L(c, \underline{g}, G) = \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} G_{ij}^2 - \sum_{k=1}^{m} \lambda_k \left[c + (\underline{x}_k^+ - \underline{x}_0)^T \underline{g} + \frac{1}{2}(\underline{x}_k^+ - \underline{x}_0)^T G(\underline{x}_k^+ - \underline{x}_0) \right], \quad (3.7)$$

with respect to the parameters of L is zero. This implies that

$$\sum_{k=1}^{m} \lambda_k = 0, \qquad \sum_{k=1}^{m} \lambda_k (\underline{x}_k^+ - \underline{x}_0) = 0, \quad (3.8)$$

and

$$G = \sum_{k=1}^{m} \lambda_k (\underline{x}_k^+ - \underline{x}_0)(\underline{x}_k^+ - \underline{x}_0)^T. \quad (3.9)$$

By substituting (3.9) into (3.4), we obtain

$$D(\underline{x}^+) = \frac{1}{2} \sum \lambda_k \left[(\underline{x}^+ - \underline{x}_0)^T (\underline{x}_k^+ - \underline{x}_0) \right]^2 + c + (\underline{x}^+ - \underline{x}_0)^T \underline{g}. \quad (3.10)$$

Equation (3.8) and the conditions (3.6) give rise to the variational square linear system

$$W \begin{pmatrix} \underline{\lambda} \\ \hline c \\ \underline{g} \end{pmatrix} = \left(\begin{array}{c|c} \Lambda & X^T \\ \hline X & O \end{array} \right) \begin{pmatrix} \underline{\lambda} \\ \hline c \\ \underline{g} \end{pmatrix} = \begin{pmatrix} \underline{r} \\ \hline \underline{0} \end{pmatrix}, \quad (3.11)$$

where W is $(m+n+1)\times(m+n+1)$ matrix, Λ has the elements

$$\Lambda_{ij}=\frac{1}{2}\left[(\underline{x}_i^+-\underline{x}_0)^T(\underline{x}_j^+-\underline{x}_0)\right]^2,\ i,j=1,2,\ldots,m, \tag{3.12}$$

X is the $(n+1)\times m$ matrix

$$\begin{pmatrix} 1 & 1 & \ldots & 1 \\ \underline{x}_1^+-\underline{x}_0 & \underline{x}_2^+-\underline{x}_0 & \ldots & \underline{x}_m^+-\underline{x}_0 \end{pmatrix}, \tag{3.13}$$

and where $\underline{r}$ is a vector in $\mathbb{R}^m$, which has the components $f(\underline{x}_i^+)-Q_{old}(\underline{x}_i^+),\ i=1,2,\ldots,m$. If we choose the points $\underline{x}_i^+,\ i=1,2,\ldots,m$, such that the matrix X has a full row rank, then the solution of the system (3.11) gives the parameters of D, and the new model is given by

$$Q_{new}(\underline{x})=Q_{old}(\underline{x})+D(\underline{x}). \tag{3.14}$$

The $(m+n+1)\times(m+n+1)$ matrix W, mentioned in the previous paragraph, depends only on the vector $\underline{x}_0$ and the vectors $\underline{x}_i^+,\ i=1,2,\ldots,m$, therefore the same occurs if the old quadratic model Q_{old} is identically zero. We begin by studing this case, and for the moment we simplify the notation by dropping the superscripts "+" which appears in the variational problem. We seek the quadratic polynomial $Q(\underline{x})=c+(\underline{x}-\underline{x}_0)^T\underline{g}+\frac{1}{2}(\underline{x}-\underline{x}_0)^TG(\underline{x}-\underline{x}_0),\ \underline{x}\in\mathbb{R}^n$, whose second derivative $G=\nabla^2Q$ has a least Frobenius norm, subject to the constraints $Q(\underline{x}_i)=f(\underline{x}_i),\ i=1,2,\ldots,m$. The vector $\underline{x}_0$, the interpolation points $\underline{x}_i,\ i=1,2,\ldots,m$, are data. The positions of the interpolatin points $\underline{x}_i,\ i=1,2,\ldots,m$, are required to have the properties:

- **A1:** Let M be the space of quadratic polynomials from $\mathbb{R}^n$ to $\mathbb{R}$ that are zero at $\underline{x}_i,\ i=1,2,\ldots,m$. Then the dimension of M is $s-m$, where $s=\frac{1}{2}(n+1)(n+2)$.
- **A2:** If $q(\underline{x}),\ \underline{x}\in\mathbb{R}^n$, is any linear polynomial that is zero at $\underline{x}_i,\ i=1,2,\ldots,m$, then q is identically zero.

Condition **A1** implies that the conditions $Q(\underline{x}_i)=f(\underline{x}_i),\ i=1,2,\ldots,m$, are consistants, so we can choose a quadratic model Q_0 that can satisfy them, hence the required Q has the form

$$Q(\underline{x})=Q_0(\underline{x})-r(\underline{x}),\ \underline{x}\in\mathbb{R}^n, \tag{3.15}$$

where $r(\underline{x})$ is an element of M that gives the least value of the Frobenius norm $\| \nabla^2 Q_0 - \nabla^2 r \|_F$. This condition provides a unique matrix $\nabla^2 r$. Moreover, if two different functions $r \in M$ have the same second derivative matrix, then the difference between them is a nonzero linear polynomial, which is not allowed by condition **A2**. Therefore the given variational problem (3.11) has a unique solution.

When the derivatives are exist, in the case of unconstrained optimization. The problem of the minimization of the Frobenius norm of the change to the second derivative matrix of a quadratic, is solved by a well-known algorithm definded by (Fletcher, 1987). The problem satisfies $Q_{k+1}(\underline{x}_{opt}^+) = Q_{new}(\underline{x}_{opt}^+) = f(\underline{x}_{opt}^+)$ and $\nabla Q_{new}(\underline{x}_{opt}^+) = \nabla f(\underline{x}_{opt}^+)$, where $\underline{x}_{opt}^+$ is given by:

$$\underline{x}_{opt}^+ = \begin{cases} \underline{x}_{opt} + \underline{p}_k \ , & f(\underline{x}_{opt} + \underline{p}_k) < f(\underline{x}_{opt}) \\ \underline{x}_{opt}, & f(\underline{x}_{opt} + \underline{p}_k) \geq f(\underline{x}_{opt}), \end{cases}$$

where $\underline{p}_k$ minimizes:

$$Q_{old}(\underline{x}_{opt} + \underline{p}_k), \text{ subject to } \| \underline{p}_k \| \leq \Delta_k.$$

So Q_{k+1} has the form

$$Q_{new}(\underline{x}_{opt}^+ + \underline{p}_k) = f(\underline{x}_{opt}^+) + \underline{p}_k^T \nabla f(\underline{x}_{opt}^+) + \frac{1}{2} \underline{p}_k^T B_{k+1} \underline{p}_k, \ \underline{x} \in \mathbb{R}^n, \tag{3.16}$$

for some $n \times n$ symmetric matrix $B_{k+1} = B_{new}$ which is an approximation of $\nabla^2 f(\underline{x}_{opt} + \underline{p}_k)$. The usual choice of B_{new} depends on the obseration that, if f is twice differentiable, then its second derivatives have the property

$$\left\{ \int_0^1 \nabla^2 f(\underline{x}_{opt} + \theta \underline{p}_k) d\theta \right\} \underline{p}_k = \nabla f(\underline{x}_{opt} + \underline{p}_k) - \nabla f(\underline{x}_{opt}). \tag{3.17}$$

We see that the left hand side is the product $\nabla^2 f \underline{p}_k$ if f happens to be a quadratic polynomial, where $\nabla^2 f$ is the constant second derivative matrix of f. Therefore, the difference between the gradients on the right hand side of (3.17) is calculated, and B_{new} is required to satisfy the condition

$$B_{new} \underline{p}_k = \nabla f(\underline{x}_{opt} + \underline{p}_k) - \nabla f(\underline{x}_{opt}). \tag{3.18}$$

A good discussion this technique can be found in (Dennis and Schabel, 1983). Equation (3.18) provides only n constraints on the new model (3.16). Because B_{new} is an $n \times n$ symmetric matrix, the number of the remaining degrees of freedom is $\frac{1}{2}n(n-1)$, which are fixed by different methods in different ways. In particular, the well-known symmetric Broyden formula is relevant in this respect. It takes up the freedom by minimizing the Frobenius norm

$$\| B_{new} - B_k \|_F = \| \nabla^2 Q_{new} - \nabla^2 Q_{old} \|_F, \tag{3.19}$$

of the difference between the second derivative matrices of the old and new quadratic model. Thus, the symmetric Broyden formula enjoys the highly useful property that is given in theorem 3.2.1 below. The theorem is presented in a way that is also applicable to the methods of optimization without derivatives. Specifically, we write Q_{new} in the form

$$Q_{new}(\underline{x}) = \sum_{j=1}^{s} \mu_j \ell_j(\underline{x}), \ \underline{x} \in \mathbb{R}^n, \tag{3.20}$$

where $\ell_j, \ j = 1, 2, \ldots, s$, is the basis of the linear space of polynomials of degree at most two from $\mathbb{R}^n$ to $\mathbb{R}$, the value of s being $\frac{1}{2}(n+1)(n+2)$. The condition (3.18) on $B_{old} = \nabla^2 Q_{new}$ provides n linear constraints on the coefficients $\mu_j, j = 1, 2, \ldots, s$. Further, the properties

$$Q_{new}(\underline{x}^+_{opt}) = f(\underline{x}^+_{opt}) \text{ and } \nabla Q_{new}(\underline{x}^+_{opt}) = \nabla f(\underline{x}^+_{opt}), \tag{3.21}$$

are aslo linear equality constraints on the coefficients of the expression (3.20).
We let M be the set of polynomials of the form (3.20) that satisfy these consraints, thus M is a linear manifold. It follows from equations (3.17) and (3.21), that if f happens to be a quadratic polynomial, then f is also an element of M.

Theorem 3.2.1. *Let $f(\underline{x}), \ \underline{x} \in \mathbb{R}^n$, be a quadratic polynomial, and let the old quadratic polynomial model Q_{old} be available. Let new model Q_{new} be in the form of equation (3.20), and satislying the linear constraint (3.31). If the degrees of freedom in Q_{new} is taken by the Frobenius norm $\| \nabla^2 Q_{new} - \nabla^2 Q_{old} \|_F$, then this construction provides the relation*

$$\| \nabla^2 Q_{new} - \nabla^2 f \|_F^2 = \| \nabla^2 Q_{old} - \nabla^2 f \|_F^2 - \| \nabla^2 Q_{new} - \nabla^2 Q_{old} \|_F^2 \leq \| \nabla^2 Q_{old} - \nabla^2 f \|_F^2 . \tag{3.22}$$

Proof: let θ be any real number, consider the function $\{Q_{new}(\underline{x}) - Q_{old}(\underline{x})\} + \theta\{f(\underline{x}) - Q_{new}(\underline{x})\}, \ \underline{x} \in \mathbb{R}^n$. It is a quadratic polynomial and its values at $\underline{x}_i, \ i = 1, 2, \ldots, m$ are indepedent of θ, because of the coditions (3.3) on Q_{new}. It follows from the construction of $Q_{new} - Q_{old}$ that the least value of the Frobenius norm $\| (\nabla^2 Q_{new} - \nabla^2 Q_{old} + \theta(\nabla^2 f - \nabla^2 Q_{new}) \|_F, \theta \in R$, occurs when θ is zero, which implies the equation

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\left\{(\nabla^2 Q_{new}(\underline{x}))_{ij} - (\nabla^2 Q_{old}(\underline{x}))_{ij}\right\}\left\{(\nabla^2 f(\underline{x}))_{ij} - (\nabla^2 Q_{new}(\underline{x}))_{ij}\right\} = 0.$$

We see that the left hand side of this identity is half the difference between the right and left hand sides of expression (3.22). Therefore, the properties (3.22) is achieved.□

The theorem implies the inequality

$$\| \nabla^2 Q_{new} - \nabla^2 f \|_F \leq \| \nabla^2 Q_{old} - \nabla^2 f \|_F, \tag{3.23}$$

when f is quadratic. Thus, if the symmetric Broyden formula is employed on every iteration, then the Frobenius norm of the error of the approximation $\nabla^2 Q_{old} \approx \nabla^2 f$ decreases monotonically during the iterative procedure. This property is welcoming, but at first sight does not cause enthusiasm, because $\| \nabla^2 Q_{old} - \nabla^2 f \|_F$ may stay bounded away from zero as $k \longrightarrow \infty$. On the other hand, the theorem aslo provides the limit

$$\lim_{k \longrightarrow \infty} \| \nabla^2 Q_{new} - \nabla^2 Q_{old} \|_F = 0, \tag{3.24}$$

which causes $\underline{x}_k, \ k = 1, 2, \ldots,$ to converge to the optimal vector at a superlinear rate, when the objective function is quadratic, for many of the usual choice of the changes $\underline{p}_k$ to the variables. These changes are derived from the quadratic models, and then the standard way of proving superlinear convergence, due (Broyden et al., 1973), requires the condition

$$\lim_{k \longrightarrow \infty} \| \nabla f(\underline{x}_{opt} + \underline{p}_k) - \nabla Q_{old}(\underline{x}_{opt} + \underline{p}_k) \| \ / \ \| \underline{p}_k \| = 0. \tag{3.25}$$

In other words, as k becomes large, the approximation $Q(\underline{x}) \approx f(\underline{x})$ has to provide a good prediction of the new gradient $\nabla f(\underline{x}_{opt} + \underline{p}_k)$. Now the property $\nabla Q_{old}(\underline{x}_{opt}) = \nabla f(\underline{x}_{opt})$, the constraint (3.17) on $B_{new} = \nabla^2 Q_{new}$ and $Q_{old} \in M$ imply the identity

$$\| \nabla f(\underline{x}_{opt} + \underline{p}_k) - \nabla Q_{old}(\underline{x}_{opt} + \underline{p}_k) \| \ / \ \| \underline{p}_k \| =$$

$$\| \left\{ \nabla f(\underline{x}_{opt} + \underline{p}_k) - \nabla f(\underline{x}_{opt}) \right\} - \left\{ \nabla Q_{old}(\underline{x}_{opt} + \underline{p}_k) - \nabla Q_{old}(\underline{x}_{opt}) \right\} \| \; / \; \| \; \underline{p}_k \; \| =$$
$$\| \; (\nabla^2 Q_{new} - \nabla^2 Q_{old}) \underline{p}_k \; \| \; / \; \| \; \underline{p}_k \; \| \; . \tag{3.26}$$

It follows from equation (3.24) that the limit (3.25) is achieved. This argument is also valid for many other objective functions that have continuous second derivatives, because then equation (3.22) holds if $\nabla^2 f$ is the matrix inside the braces of expression (3.17), but careful attension has to be given to the changes to $\nabla^2 f$ that occur in expression (3.22) during the sequence of iterations. The purpose of these remarks is to point out that the Frobenius norm method for updating quadratic models may provide fast convergence, even if $\| \; \nabla^2 Q_{old} - \nabla^2 f \; \|_F$ does not become small as k increased. For more details see (Powell, 2003).

3.3 The Lagrange functions of the interpolation points

The Lagrange functions of the interpolation points $\underline{x}_i, \; i = 1, 2, \ldots, m$, are quadratic polynomials $\ell_j(\underline{x}), \; \underline{x} \in \mathbb{R}^n, \; j = 1, \; 2, \ldots, m$, that satisfy the conditions

$$\ell_j(\underline{x}_i) = \delta_{ij}, \; 1 \leq i, j \leq m, \tag{3.27}$$

where δ_{ij} the kronecker delta. Lagrange polynomials play a fundamental role in preserving nonsingularity of the system (3.11) of the previous section. In order for these polynomials to be applicable to the variational system (3.11) of section (3.2), we retain the conditions **A1** and **A2** (of section (3,2)) on the positions of the interpolation points, and for each $j = 1 \; , 2, \ldots, m$, we take up the freedom in $\ell_j(\underline{x})$ by minimizing the Frobenius norm $\| \; \nabla^2 l_j \; \|_F$ subject to the consraints in (3.27), therefore, the parameters of ℓ_j are defined by the system (3.11), if we replace the right hand side of this system by the j-th coordinate vector in $\mathbb{R}^{m+n+1}$. Thus, if we let Q be the quadratic polynomial

$$Q(\underline{x}) = \sum_{j=1}^{m} f(\underline{x}_j) \ell_j(\underline{x}), \quad \underline{x} \in \mathbb{R}^n, \tag{3.28}$$

then its parameters satisfy equation (3.11). It follows from the nonsingularity of this system, that expression (3.28) is the Lagrange form of the solution of the variational

problem of section (3.2). Let H be the inverse of the matrix W of the system (3.11) of section (3.2). The definition of ℓ_j, where j is any integer in $[1, m]$, implies that the j-th column of H provides the parameters of ℓ_j. In particular, because of equation (3.9), ℓ_j has the second derivative matrix

$$G_j = \nabla^2 \ell_j = \sum_{k=1}^{m} H_{kj}(\underline{x}_k - \underline{x}_0)(\underline{x}_k - \underline{x}_0)^T, j = 1, 2, \ldots, m. \tag{3.29}$$

Further, c_j is equal to H_{m+1j} and $\underline{g}$ is equal to the vector with component $H_{ij}, i = m+2, m+3, \ldots, m+n+1$. We find that ℓ_j is the polynomial

$$\ell_j(\underline{x}) = c_j + (\underline{x} - \underline{x}_0)^T \underline{g}_j + \frac{1}{2}(\underline{x} - \underline{x}_0)^T G_j (\underline{x} - \underline{x}_0), \underline{x} \in \mathbb{R}^n. \tag{3.30}$$

Because the parameters of $\ell_j(\underline{x})$ depend on H, we require the elements of the matrix H to be available, but there is no need to store the matrix W. We only store one column of this matrix.

To discuss the relation between the polynomial $\ell_j(\underline{x})$ and nonsingularity of the system (3.11), let $\underline{x}^+$ be the new vector of variables that will replace one of the interpolation points $\underline{x}_i,\ i = 1, 2, \ldots, m$. When $\underline{x}^+$ replaces $\underline{x}_t,\ t \in \{1,\ 2, \ldots, m\}$, $\underline{x}_t$ will be dismissed, so the new interpolation points are the vectors

$$\underline{x}_t^+ = \underline{x}^+, \quad \underline{x}_i^+ = \underline{x}_i, \quad i \in \{1, 2, \ldots, m\} \setminus \{t\}\,. \tag{3.31}$$

One advantage of the Lagrange functions, is that they provide a convenient way of maintaining the conditions **A1** and **A2** of section (3.2). Indeed, it is shown below that these conditions are inherited by the new interoplation points if t is chosen so that $\ell_t(\underline{x}^+)$ is nonzero. At least one of the values $\ell_j(\underline{x}^+), j = 1, 2, \ldots, m$, is nonzero, because interpolation to a constant function yields

$$\sum_{j=1}^{m} \ell_j(\underline{x}) = 1, \quad \underline{x} \in \mathbb{R}^n. \tag{3.32}$$

Let $\ell_t(\underline{x}^+)$ be nonzero, let the condition **A1** be satisfied, let M^+ be the space of quadratic polynomials from $\mathbb{R}^n$ to $\mathbb{R}$ that are zero at $\underline{x}_i^+,\ i = 1, 2, \ldots, m$. We have to prove that the dimension of M^+ is $s - m$.

We employ the linear space, M^- say, of quadratic polynomials that are zero at $\underline{x}_i^+ =$

$\underline{x}_i, \quad i \in \{1,\ 2, \ldots, m\} \setminus \{t\}$. It follows from **A1** that the dimension of M^- is $s - m + 1$. Further the dimension of M^+ is $s - m$ if and only if an element of M^- is nonzero at $\underline{x}_t^+ = \underline{x}^+$. The Lagrange equation (3.27) shows that ℓ_t is in M^-. Therefore, the property $\ell_t(\underline{x}^+) \neq 0$ gives the result.

We now consider condition **A2**. It is achieved by the new interpolation points if the values $q(\underline{x}_i) = 0, \quad i \in \{1, 2, \ldots, m\} \setminus \{t\}$, where q is a linear polynomial, implies $q = 0$. Otherwise, we let q be a nonzero polynomial of this kind, and we deduce from **A2** that $q(\underline{x}_t)$ is nonzero. Therefore, because all the second derivatives of q are zero, the function $q(\underline{x})/q(\underline{x}_t), \underline{x} \in \mathbb{R}^n$ is the Lagrange function ℓ_t.Thus, if q is a nonzero linear polynomial that takes the values $q(\underline{x}_i) = 0, \quad i \in \{1, 2, \ldots, m\} \setminus \{t\}$, then it is a multiple of ℓ_t. Such a polynomial cannot vanish at $\underline{x}_t^+$ because of the assumption $\ell_t(\underline{x}^+) \neq 0$. It follows that condition **A2** is also inherited by the new interpolation points (Powell, 1998, 2001).

These remarks suggest that, in the presence of computer rounding error, the preservation of conditions **A1** and **A2** by a sequence of iterations may be more stable if $\mid \ell_t(\underline{x}^+) \mid$ is relatively large. If we want to improve the accuracy of the quadratic model, we select t, the index of the interpolation point $\underline{x}_t$ that is going to be replaced by $\underline{x}^+$, before the point $\underline{x}^+$. Indeed, $\underline{x}_t$ is often an element of the set $\{\underline{x}_i,\ i = 1, 2, \ldots, m\}$ that is furthest from the best $\underline{x}_{opt}$, because Q intended to be an adequate approximation to f with the trust-region subproblem. Therefore, we pick the index t, such that the value of $\mid \ell_t(\underline{x}^+) \mid$ is made to be relatively large, by setting $\underline{x}^+$ to be an estimate of the vector $\underline{x} \in \mathbb{R}^n$, that solves the alternative subproblem

$$\max \mid \ell_t(\underline{x}) \mid, \quad \text{subject to:} \quad \| \underline{x} - \underline{x}_{opt} \| \leq \Delta_k, \tag{3.33}$$

so again the availability of the Lagrange function is required.

We recall from section (3.2) that the new model Q_{new} is formed by adding the difference $D(\underline{x})$ to Q_{old} where $D(\underline{x}) = Q_{new} - Q_{old}$ is the quadratic model whose second derivative has the least Frobenius norm subject to the constraints (3.6). As we mentioned in this section, the parameters of $\ell_t(\underline{x})$ are define by the system (3.11), if we replace the right hand side of this system by t-th coordinate vector in $\in \mathbb{R}^n$. Equation (3.27) imply that only the t-th right hand side of these conditions can be nonzero. Therefore, by considering

the Lagrange form (3.28) of the solution of the variational problem of section (3.2), we deduce that $Q_{new} - Q_{old}$ is a multiple of the t-th Lagrange function, ℓ_t^+ say, for the new interpolation points, where the multiplying factor is defined by the constraint (3.6) in the case $i = t$. Thus, Q_{new} is the quadratic polynomial

$$Q_{new} = Q_{old} + \left\{f(\underline{x}^+) - Q_{old}(\underline{x}^+)\right\} \ell_t(\underline{x}^+),\ \underline{x} \in \mathbb{R}^n. \tag{3.34}$$

Also it follows that the constant term c^+ and the component of the vector $\underline{g}^+$ of the new model are

$$\begin{aligned} c^+ &= c_{old} + \left\{f(\underline{x}^+) - Q_{old}(\underline{x}^+)\right\} H^+_{m+1t} \\ \underline{g}^+ &= \underline{g}_{old} + \left\{f(\underline{x}^+) - Q_{old}(\underline{x}^+)\right\} H^+_{m+j+1t},\ j = 1, 2, \ldots, n, \end{aligned} \tag{3.35}$$

and the second derivative matrix $G^+ = \nabla^2 Q$ is given by

$$\begin{aligned} G^+ &= G_{old} + \left\{f(\underline{x}^+) - Q_{old}(\underline{x}^+)\right\} \nabla^2 \ell_t(\underline{x}^+) \\ &= G_{old} + \left\{f(\underline{x}^+) - Q_{old}(\underline{x}^+)\right\} \sum_{k=1}^{m} H^+_{kt}(\underline{x}^+_k - \underline{x}_0)(\underline{x}^+_k - \underline{x}_0)^T. \end{aligned} \tag{3.36}$$

We see that G^+ can be constructed by adding m matrices of rank one to G_{old}, but the work of that task would be $O(mn^2)$, which is unwelcome in the case $m = O(n)$, because we are trying to complete the updating in only $O(m^2)$ operations. Therefore, instead of storing G_{old} explicitly, we apply the form

$$G_{old} = \Gamma + \sum_{k=1}^{m} \gamma_k (\underline{x}_k - \underline{x}_0)(\underline{x}_k - \underline{x}_0)^T, \tag{3.37}$$

which defines the matrix Γ for any choice of $\gamma_k,\ k = 1, 2, \ldots, m$, these multipliers being stored. We seek a similar expression for G^+. Specifically, because of the change (3.31) to the positions of the interpolation points, we let Γ^+ and G^+ be the matrices

$$\Gamma^+ = \Gamma + \gamma_t (\underline{x}_t - \underline{x}_0)(\underline{x}_t - \underline{x}_0)^T, \tag{3.38}$$

$$G^+ = \Gamma^+ + \sum_{k=1}^{m} \gamma_k^+ (\underline{x}^+_k - \underline{x}_0)(\underline{x}^+_k - \underline{x}_0)^T \tag{3.39}$$

Then equations (3.36) and (3.37) provide the values

$$\gamma_k^+ = \gamma_k(1 - \delta_{kt}) + \left\{f(\underline{x}^+) - Q_{old}(\underline{x}^+)\right\} H^+_{kt},\ k = 1, 2, \ldots, m. \tag{3.40}$$

The quadratic model of the first iteration is calculated from the interpolation conditions $f(\underline{x}_i) = Q(\underline{x}_i)$ by solving the variational problem of section (3.2). Therefore, because of equation (3.4), the choice $\Gamma = O$ and $\gamma_k = \lambda_k,\ k = 1, 2, \ldots, m$ can be made initially for the second derivative matrix (3.37).

3.4 The procedure for updating the matrix H

We introduce the calculation of H^+ from H by indentifying the stable property that is achieved. We recall that the change (3.31) to the interpolation points causes the symmetric matrices $W = H^{-1}$ and $W^+ = (H^+)^{-1}$ to differ only in their t-th row and colmun. Also, we recall that W is not stored. Therefore, our formula of H^+ is going to depend only on H and the vector $\underline{w}_t^+ \in \mathbb{R}^{m+n+1}$, which is the t-th column of W^+. These data completly define H^+, because in theory the updating calculation can be done by inverting H to give W. Then the availability of $\underline{w}_t^+$ allow the symmetric matrix W^+ to be formed from W, then H^+ is set to the inverse of W^+. This procedure provides excellent protection against the accumulation of computer rounding errors. Two formulae of H^+ are presented. The first one can be displayed as:

$$H^+ = H + \frac{1}{\sigma_t^+}\left[\alpha^+(\underline{e}_t - H\underline{w}_t^+)(\underline{e}_t - H\underline{w}_t^+)^T - \beta_t^+ H\underline{e}_t\underline{e}_t^T H\right] + \frac{1}{\sigma^+}\left[\tau_t^+\left\{H\underline{e}_t(\underline{e}_t - H\underline{w}_t^+)^T + (\underline{e}_t - H\underline{w}_t^+)\underline{e}_t^T H\right\}\right], \tag{3.41}$$

where $\underline{e}_t$ is the t-th coordinate vector in $\mathbb{R}^{m+n+1}$,

$$\alpha_t^+ = \underline{e}_t^T H\underline{e}_t, \quad \beta_t^+ = (\underline{e}_t - H\underline{w}_t^+)^T\underline{w}_t^+,$$

$$\tau_t^+ = \underline{e}_t^T H\underline{w}_t^+, \quad \text{and} \quad \sigma_t^+ = \alpha_t^+\beta_t^+ + (\tau_t^+)^2. \tag{3.42}$$

Theorem 3.4.1. *If H is nonsingular and symmetric, and if σ_t is nonzero, then the expressions (3.41) and (3.42) provide the matrix H^+ that satisfies equation (3.41).*

Proof: see appendix A.

The vector $\underline{w}_t^+$ in (3.41) is the t-th column of the matrix W^+ of the system (3.11)

for the new interpolation points. Therefore, because the choice $\underline{x}_t^+ = \underline{x}^+$, $\underline{w}_t^+$ has the components

$$(\underline{w}_t^+)_i = \frac{1}{2}\left\{(\underline{x}_i^+ - \underline{x}_0)^T(\underline{x}^+ - \underline{x}_0)\right\}^2, \quad i = 1, 2, \ldots, m,$$

$$(\underline{w}_t^+)_{m+1} = 1, \quad \text{and} \quad (\underline{w}_t^+)_{m+i+1} = (\underline{x}^+ - \underline{x}_0)_i, \quad i = 1, 2, \ldots, n. \tag{3.43}$$

Moreover, let $\underline{w} \in \mathbb{R}^{m+n+1}$ be the vector that have the component

$$w_i = \frac{1}{2}\left\{(\underline{x}_i - \underline{x}_0)^T(\underline{x}^+ - \underline{x}_0)\right\}^2, \quad i = 1, 2, \ldots, m,$$

$$w_{m+1} = 1, \quad \text{and} \quad w_{m+i+1} = (\underline{x}^+ - \underline{x}_0)_i, \quad i = 1, 2, \ldots, n. \tag{3.44}$$

It follows from the positions (3.31) of the new interpolation points that $\underline{w}_t^+$ is the sum

$$\underline{w}_t^+ = \underline{w} + \eta_t \underline{e}_t. \tag{3.45}$$

where $\underline{e}_t$ is the t-th coordinate vector in $\mathbb{R}^{m+n+1}$, and η_t is the difference

$$\eta_t = \underline{e}_t^T \underline{w}_t^+ - \underline{e}_t^T \underline{w} = \frac{1}{2} \parallel \underline{x}^+ - \underline{x}_0 \parallel^4 - \underline{e}_t^T \underline{w}. \tag{3.46}$$

An advantage of working with $\underline{w}$ instead of $\underline{w}_t^+$ is that, if $\underline{x}^+$ is available before t is selected, which happens when $\underline{x}^+$ is calculated from trust-region subproblem, then $\underline{w}$ is independent of t. Therefore, we derive a new version of the updating formula (3.41) by making the substitution (3.46). Specifically, we replace $\underline{e}_t - H\underline{w}_t^+$ by $\underline{e}_t - H\underline{w} - \eta_t H\underline{e}_t$ in equation (3.41). Thus H^+ is given by

$$H^+ = H + \frac{1}{\sigma_t}\left[\alpha_t(\underline{e}_t - H\underline{w})(\underline{e}_t - H\underline{w})^T - \beta_t H\underline{e}_t\underline{e}_t^T H + \tau_t\left\{H\underline{e}_t(\underline{e}_t - H\underline{w})^T + (\underline{e}_t - H\underline{w})\underline{e}_t^T H\right\}\right], \tag{3.47}$$

where the parameters of (3.47) have the values

$$\alpha_t = \underline{e}_t^T H \underline{e}_t, \qquad \beta_t = (\underline{e}_t - H\underline{w})^T \underline{w} + \eta_t,$$

$$\tau_t = \underline{e}_t^T H \underline{w}, \qquad \text{and} \qquad \sigma_t = \alpha_t\beta_t + \tau_t^2. \tag{3.48}$$

Another advatage of working with $\underline{w}$ instead of $\underline{w}_t^+$ in the updating procedure is that the first m components of the product $H\underline{w}$ are the values $\ell_j(\underline{x}^+), j = 1, 2, \ldots, m$, of the current Lagrange functions at the new point $\underline{x}^+$. We justify this assertion by recalling

equations (3.29) and (3.30), and the observation that the elements H_{m+1j} and $H_{ij},\ i = m+2,\ldots,m+n+1$, are c_j and the components of $\underline{g}_j$, respectively, where j is any integer from $[1,m]$. Specifically, by substituting the matrix (3.29) into equation (3.30), we find that

$$\ell_j(\underline{x}^+) = H_{m+1j} + \sum_{i=1}^{n} H_{m+i+1j}(\underline{x}^+ - \underline{x}_0) + \frac{1}{2}\sum_{i=1}^{m} H_{ij}\left\{(\underline{x}_i - \underline{x}_0)^T(\underline{x}^+ - \underline{x}_0)\right\}^2, \quad (3.49)$$

which is analogous to the form (3.30). Hence, because of the choice (3.44) of the components of $\underline{w}$, the symmetric H gives the required result

$$\ell_j(\underline{x}^+) = \sum_{i=1}^{m+n+1} H_{ij}\underline{w}_i = e_j^T H\underline{w}, \quad j = 1,2,\ldots,m. \quad (3.50)$$

Moreover, some cancellations occur if we combine η_t and β_t, and the parameters of the updating formula (3.47) takes the values

$$\alpha_t = \underline{e}_t^T H\underline{e}_t = H_{tt}, \qquad \beta_t = \frac{1}{2}\parallel \underline{x}^+ - \underline{x}_0 \parallel^4 - \underline{w}^T H\underline{w},$$

$$\tau_t = \ell_t(\underline{x}^+) = \underline{e}_t^T H\underline{w}, \qquad \text{and} \qquad \sigma_t = \alpha_t\beta_t + \tau_t^2. \quad (3.51)$$

The results (3.51) not only useful in practice, but also relevent to the nearness of the matrix $W^+ = (H^+)^{-1}$ to be singular. Indeed, the formula (3.47) suggests that difficulties may occur from the large elements of H^+ if $\mid \sigma \mid$ is unusually small. Further, we recall from section (3.3) that we avoid singularity in W^+ by choosing t so that $\ell_t(\underline{x}^+) = \tau$ is nonzero.

3.5 The NEW Unconstrained Opimination Algorithm (NEWUOA)

In this section we give an outline discusstion of NEWUOA (Powell, 2006) algorithm, because our algorithm which is presented in chapter 4 is an extension of NEWUOA algorithm. The NEWUOA algorithm is an unconstrained derivative-free algorithm, the name NEWUOA is an acronym for NEW Unconstrained Opimination Algorithm. NEWUOA

algorithm seeks the least value of a function $f(\underline{x}),\ \underline{x} \in \mathbb{R}^n$ where $f(\underline{x})$ can be caculated for any vector of variables, but the derivatives of $f(\underline{x})$ are not available. The method of NEWUOA is iterative, k and n are reserved for the iteration number and the number of variables, respectively. NEWUOA algorithm tries to construct a suitable quadratic model from few data. The model $Q(\underline{x}),\ \underline{x} \in \mathbb{R}^n$ at the beginning of a typical iteration, has to satisfy only m interpolation conditions

$$Q(\underline{x}_i) = f(\underline{x}_i),\ i = 1, 2, \ldots, m, \tag{3.52}$$

where $f(\underline{x})$ is the objective function, and where m is prescribed by the user, and where the positions of the different points $\underline{x}_i,\ i = 1, 2, \ldots, m$, are generated automatically. It is required that $m \geq n+2$ in order that equation (3.52) always provides some conditions on the second derivative matrix $\nabla^2 Q$, and require $m \leq \frac{1}{2}(n+1)(n+2)$, because otherwise no quadratic model Q can satisfy all the equations for a general right hand side. Numerical results support the choice $m = 2n + 1$. The success of NEWUOA algorithm is due to the technique discussed in section (3.2). Given an old model $Q_{old}(\underline{x})$, let the new model $Q_{new}(\underline{x})$ be required to satisfy some conditions that are compatible and leave some freedom in the parameters of $Q_{new}(\underline{x})$. The algorithm takes up this freedom by minimizing $\| \nabla^2 Q_{new} - \nabla^2 Q_{old} \|_F$. The conditions on the new model $Q_{new}(\underline{x})$ are the interpolation conditions (3.52). Thus $\nabla^2 Q_{new}(\underline{x})$ is defined uniquely and $Q_{new}(\underline{x})$ itself is unique, as proved in section 3.2, provided that the rows of the $(n + 1) \times m$ matrix X of equation (3.12), are linearly independent.

Let

$$H = \left(\begin{array}{c|c} \Omega & \Xi^T \\ \hline \Xi & \Upsilon \end{array}\right) = W^{-1}, \tag{3.53}$$

where W is the matrix of the system (3.11). In theory, the rank of Ω, which is the leading $m \times m$ submatrix of H, is only $m - n - 1$, but this property is lost in practice, because of rounding errors. Therefore, at each iteration, NEWUOA starts a factorization of Ω is stored instead of Ω itself, when updating H.

The user of NEWUOA algorithm has to define the objective function by a subroutine that computes $f(\underline{x})$ for any vector of variable, $\underline{x} \in \mathbb{R}^n$. An initial vector $\underline{x}_0 \in \mathbb{R}^n$, the number m of the interpolation points (3.52), and the initial and final trust-region radius, namely

ρ_{beg} and ρ_{end} are required too. For each iteration k, the parameter ρ_k, which is the lower bound on the trust-region radius, is also required. As mentioned, an integer m satisfying

$$n+2 \leq m \leq \frac{1}{2}(n+1)(n+2), \tag{3.54}$$

and that often the choice $m = 2n+1$ is good for efficiency. The initial interpolation points $\underline{x}_i, i = 1, 2, \ldots, m$, include $\underline{x}_0$, while the other points have the property $\| \underline{x}_i - \underline{x}_0 \|_\infty = \rho_{beg}$.

Let $\underline{x}_{opt}$ be an interpolation point such that $f(\underline{x}_{opt})$ is the least calculated value of $f(\underline{x})$ so far. Let the quadratic model at the beginning of the k-th iteration be the function

$$Q_k(\underline{x}_{opt} + \underline{p}_k) = f(\underline{x}_{opt}) + \underline{p}^T g_k + \frac{1}{2}\underline{p}_k^T G_k \underline{p}_k, \ \underline{p}_k \in \mathbb{R}^n, \tag{3.55}$$

its parameters being the vector $\underline{g}_k$, and the $n \times n$ matrix $\nabla^2 Q_k = G_k$. Let the interpolation conditions of Q be the equations (3.52). There are two types of iterations in NEWUOA, namely "trust-region" iteration and "model" iteration. On each trust-region iteration, the step $\underline{p}_k$ from $\underline{x}_{opt}$ is the vector that is calculated by the truncated conjugate gradient method to the subproblem

$$\min Q_k(\underline{x}_{opt} + \underline{p}_k), \text{ subject to } \| \underline{p}_k \| \leq \Delta_k, \tag{3.56}$$

If $\| \underline{p}_k \| < \frac{1}{2}\rho_k$ occurs, then the view is taken that $\underline{x}_{opt} + \underline{p}_k$ is close to the $\underline{x}_{opt}$, and the current iteration is therefore switched to "model" type. If $\| \underline{p}_k \| \geq \frac{1}{2}\rho_k$, then usually a new function value $f(\underline{x}_{opt} + \underline{p}_k)$ is calculated. Further, ρ_{k+1} is set ρ_k , a new trust-region radius $\Delta_{k+1} \geq \Delta_k$ is chosen in a usual way that depends on the RATIO presented in equation 2.38, i.e the RATIO:

$$\text{RATIO} = \frac{f(\underline{x}_{opt}) - f(\underline{x}_{opt} + \underline{p}_k)}{Q_k(\underline{x}_{opt}) - Q_k(\underline{x}_{opt} + \underline{p}_k)}, \tag{3.57}$$

and the new point is given by

$$\underline{x}_{opt} = \begin{cases} \underline{x}_{opt} + \underline{p}_k, & f(\underline{x}_{opt} + \underline{p}_k) < f(\underline{x}_{opt}) \\ \underline{x}_{opt}, & f(\underline{x}_{opt} + \underline{p}_k) \geq f(\underline{x}_{opt}) \end{cases}. \tag{3.58}$$

and the new model is constructed. If RATIO< 0.1, then the algorithm test whether the geometry of the interpolation points need to be improved or the the trust-region shrinked.

If the geometry of the interpolation points need to be improved, then we switch to the "model" iteration. The "model iteration" tries to improve the geometry of the interpolation set by choosing a step $\underline{p}_k$ which maximizes the Lagrange polynomial $\ell_t(\underline{x}_{opt} + \underline{p}_k)$, where t is the index of the interolation point that will be removed in order to improve the model, as discussed in section 3.3. Each "model" iteration is followed by a "trust-region" iteration. If RATIO≥ 0.1, then the next iteration will be a "trust-region" iteration, the algorithm proceeds. Many other details of NEWUOA are discussed in chapter 4, for more detials see (Powell, 2006).

3.6 The BOBYQA Algorithm

BOBYQA, an extension of NEWUOA to simple bound constraints. The name BOBYQA is an acronym for Bound Opimination BY Quadratic Approximation. BOBYQA, is an iterative algorithm for a minimum of a function $f(\underline{x})$, $\underline{x} \in \mathbb{R}^n$ subject to simple bounds $\underline{a} \leq \underline{x} \leq \underline{c}$, where $f(\underline{x})$ is specified by "a black box" that returns the value of $f(\underline{x})$ for any feasible point $\underline{x}$. No derivatives are available for $f(\underline{x})$, therefore, each iteration employs a quadratic approximation Q to $f(\underline{x})$ that satisfies the coditions $Q(\underline{x}_i) = f(\underline{x}_i)$, $i = 1, 2, \ldots, m$, where the interpolation points being chosen and adjusted automatically. The value $m = 2n + 1$ is recommended. These conditions leave $\frac{1}{2}n(n-1)$ degree of freedom in Q, taken up by minimizing the Frobenius norm of the change to the second derivative matrix of Q. Many features of NEWUOA are inherited by BOBYQA, and the difference between the two algorithms is confined to the initial calculations, the procedure that solves the subproblem

$$\min_{\underline{x} \in \mathbb{R}^n} \; Q_k(\underline{x}_{opt} + \underline{p}_k), \quad \text{subject to: } \; \underline{a} \leq \underline{x} \leq \underline{c}, \quad \| \underline{p}_k \| \leq \Delta_k, \tag{3.59}$$

, the procedure that improve the the geometry of the interpolation points and RESCUE procedure which is used to provide a better denominator σ. We will discuss these topics briefly. For more details see (Powell, 2009).

3.6.1 Initial calculations

The user of BOBYQA has to supply an initial vector of variables $\underline{x}_0 \in \mathbb{R}^n$, the vectors $\underline{a}$, $\underline{c}$, whose components satisfies $a_i \leq (\underline{x}_0)_i \leq c_i,\ i = 1, 2, \ldots, n$, the initial trust-region radius, and the number m of the interpolation conditions, where $n + 2 \leq m \leq \frac{1}{2}(n+1)(n+2)$, usually $m = 2n + 1$ is chosen. A gradient of the first quadratic model is constructed from the changes that occur in f when the steps from $\underline{x}_0$ parallel to coordinate directions are taken in $\mathbb{R}^n$, the lengths of these steps being Δ_1. When there are two steps in the same directions, they provide diagonal element of the second derivative matrix $\nabla^2 Q_1$. Because room is required for these constructions, an error is made immediately from BOBYQA if the bounds fail to satisfy the conditions $c_i \geq a_i + 2\Delta_1,\ i = 1, 2, \ldots, n$. The position of $\underline{x}_0$ must be suitable for these constructions and, if necessary, it is altered automatically to satisfy the bound conditions. Let i be an integer in the set $\{1, 2, \ldots, n\}$ and let $(\underline{x}_0)_i$, be the i-th component of the given vector $\underline{x}_0$. This component is overwritten by a_i or c_i in the case $(\underline{x}_0)_i < a_i$ or $(\underline{x}_0)_i > c_i$, respectively. Moreover, it is overwritten by $a_i + \Delta_1$ or $c_i - \Delta_1$ in the case $a_i < (\underline{x}_0)_i < a_i + \Delta_1$ or $c_i - \Delta_1 < (\underline{x}_0)_i < c_i$ respectively. In all other cases, the original value of $(\underline{x}_0)_i$ is retained. We are now ready to specify the interpolation points $\underline{x}_i,\ i = 1, 2, \ldots, m$.

We set $\underline{x}_1 = \underline{x}_0$, and, for $i = 1, 2, \ldots, n$, we define $\underline{x}_{i+1}$ and $\underline{x}_{n+i+1}$ by the formulae

$$\underline{x}_{i+1} = \underline{x}_0 + \Delta_1 \underline{e}_i, \quad \text{and} \quad \underline{x}_{n+i+1} = \underline{x}_0 - \Delta_1 \underline{e}_i,\ a_i < (\underline{x}_0)_i < c_i,$$

$$\underline{x}_{i+1} = \underline{x}_0 + \Delta_1 \underline{e}_i, \quad \text{and} \quad \underline{x}_{n+i+1} = \underline{x}_0 + 2\Delta_1 \underline{e}_i,\ (\underline{x}_0)_i = a_i,$$

$$\underline{x}_{i+1} = \underline{x}_0 - \Delta_1 \underline{e}_i, \quad \text{and} \quad \underline{x}_{n+i+1} = \underline{x}_0 - 2\Delta_1 \underline{e}_i, (\underline{x}_0)_i = c_i, \tag{3.60}$$

where $\underline{e}_i$ is the i-th unit coordinate vector in $\mathbb{R}^n$. We recall, from section 3.2, that the inverse of the matrix W of the variational problem (3.11) is employed in each iteration to assist in the calculation of the next quadratic model Q_{new} from Q_{old}. The construction of the inverse matrix H of the first iteration is given by

$$H = W^{-1} = \left(\begin{array}{c|c} \Omega & \Xi^T \\ \hline \Xi & O \end{array} \right). \tag{3.61}$$

The elements of the submatrix Ξ, and the elements of an $m \times (m-n-1)$ matrix V such that Ω is the product VV^T are defined as follows. The first row of the matrix Ξ is δ_{1j}, where $j = 1, 2, \ldots, m$. For $i = 1, 2, \ldots, n$, the $(i+1)$-th row of Ξ has exactly three nonzero elements that take the values

$$\Xi_{i+11} = -\frac{1}{\alpha_i} - \frac{1}{\beta_i}, \quad \Xi_{i+1i+1} = \frac{\beta_i}{\alpha_i(\beta_i - \alpha_i)}, \quad \text{and} \quad \Xi_{i+1n+i+1} = \frac{\alpha_i}{\beta_i(\alpha_i - \beta_i)}, \tag{3.62}$$

where α_i and β_i are assumed to be nonzero nubmers that satisfy $\alpha_i \neq \beta_i$.

For $1 \leq l \leq n$, there are exactly three nonzero entries in the l-th column of V with the values

$$V_{1l} = \frac{\sqrt{2}}{\alpha_l \beta_l}, \quad V_{l+1l} = \frac{\sqrt{2}}{\alpha_l(\beta_l - \alpha_l)}, \text{ and } V_{n+l+1l} = \frac{\sqrt{2}}{\beta_l(\alpha_l - \beta_l)}. \tag{3.63}$$

For the proof of these assertions see (Powell, 2009).

3.6.2 The solution of the trust-region subproblem

The calculation of the "trust-region" step $\underline{p}_k$ of the subproblem (3.59) is done by an active set version of the truncated conjugate gragient procedure (see section 2.7), that begins at the centre $\underline{p}_k = \underline{0}$, of the trust-region $\left\{\underline{p}_k : \| \underline{p}_k \| \leq \Delta_k\right\}$, with a restart and an enlarged active set if $\underline{p}_k$ reaches the boundary. If $\underline{p}_k$ reaches the boundary of the trust-region, then further a change may be made to $\underline{p}_k$, staying on the boundary. The alternative being terminated of the conjugate gradient iteration with $\| \underline{p}_k \| < \Delta_k$. So, there is no removal of the indices from the active set of the current subproblem. Detail of these constructions are given in (Powell, 2009).

The procedure that improves the geometry of the interpolation points employs the Lagrange polynomial of section 3.3, the only additional requirement is that the new point must satisfy the bound constraints.

3.6.3 The method of RESCUE

Occasionally, rounding errors cause severe damage to the parameters (3.48) of the formula (3.47). In particular, rounding errors may lead to a large reduction in $|\ \sigma\ |$.

Therefore, in every iteration of BOBYQA, a test is made ensures whether or not the calculated denominater $\sigma = \alpha\beta + \tau^2$ seems to be adequate. The level of tolerance in each iteration is that the updating procedure of section 3.4, proceeds as usual unless the calculated σ of (3.48) satisfies the condition

$$\sigma = \alpha\beta + \tau^2 \leq \frac{1}{2}\tau^2. \tag{3.64}$$

In case (3.64), a subroutine that has the name RESCUE, is called. It tries to provide a better denominator σ in the following way. When RESCUE is called the current H and V are rejected, the points $\underline{x}_j$ and the function values $f(\underline{x}_j),\ j = 1, 2, \ldots, m$, are retained, however, and $\underline{x}_{opt}$ still the interpolation point that has the property that $f(\underline{x}_{opt})$ is of least value of $f(\underline{x})$ so far. The current polynomial model $Q_k(\underline{x}),\ \underline{x} \in \mathbb{R}^n$ is also retained. Usaully a few of the interpolation points are replaced, which requires some new values of $f(\underline{x})$, and then Q_k is updated to interpolate the new function values. The first task of RESCUE is to shift the origin to the point $\underline{x}_0 = \underline{x}_{opt}$, and then to calculate the new points

$$\underline{\gamma}_1 = \underline{x}_0 = \underline{x}_{opt},\ \underline{\gamma}_{i+1} = \underline{x}_0 + \alpha_i \underline{e}_i \ \text{ and } \ \underline{\gamma}_{n+i+1} = \underline{x}_0 + \beta_i \underline{e}_i, \quad i = 1, 2, \ldots, n, \tag{3.65}$$

where α_i, β_i are nonzero multipliers that satisty $\alpha_i \neq \beta_i,\ i = 1, 2, \ldots, n$, and where $\underline{e}_i$ is the i-th coordinate vector in $\mathbb{R}^n$. RESCUE employs an iterative procedure that begins with the set $\left\{\underline{\hat{x}}_j = \underline{\gamma}_j,\ j = 1, 2, \ldots, m\right\}$, composed of the old point $\underline{\gamma}_1 = \underline{x}_{opt}$ and $m - 1$ new points of the form (3.65). A typical iteration of RESCUE picks an old point, $\underline{x}_l$ say, that is not in the set $\left\{\underline{\hat{x}}_j,\ j = 1, 2, \ldots, m\right\}$, and then asks the following question for $t = 1, 2, \ldots, m$. If $\underline{\hat{x}}_t$ is not one of the points $\left\{\underline{x}_j :\ j = 1, 2, \ldots, m\right\}$, how safe is it to replace $\underline{\hat{x}}_t$ by $\underline{x}_l$ in the set $\left\{\underline{\hat{x}}_j,\ j = 1, 2, \ldots, m\right\}$?. Further, the question is asked whether this choice is safe enough?. Usually the answer is an affirmative, and then the replacement of $\underline{\hat{x}}_t$ by $\underline{x}_l$ in the set $\left\{\underline{\hat{x}}_j,\ j = 1, 2, \ldots, m\right\}$ is made. Otherwise, the same question is asked for another l, which may lead to different replacements. Thus, every successful iteration of RESCUE increases the number of interpolation points in the set $\left\{\underline{\hat{x}}_j,\ j = 1, 2, \ldots, m\right\}$ by one. This procedure ends if $m - 1$ iterations are sucessful, because the set $\left\{\underline{\hat{x}}_j,\ j = 1, 2, \ldots, m\right\}$ has become the set of old interpolation points

$\{\underline{x}_j,\ j = 1, 2, \ldots, m\}$. Alternatively, the iterative procedure of RESCUE ends when a sufficient safe replacement of $\underline{\hat{x}}_j$ by $\underline{x}_l$ can not be found. The final set $\{\underline{\hat{x}}_j,\ j = 1, 2, \ldots, m\}$ is the new set of the interpolation points chosen by RESCUE, the function value $f(\underline{\hat{x}}_j)$ being calculated by RESCUE if and only if $\underline{\hat{x}}_j$ is not in the old set $\{\underline{x}_j,\ j = 1, 2, \ldots, m\}$, and then the model is updated according to the new function values $f(\underline{\hat{x}}_j)$. For more details see (Powell, 2009).

Chapter 4

Linearly Constrained Optimization by Quadratic Approximation Algorithm

4.1 Introduction

In this chapter, we describe in detail our algorithm for solving problem (1.1) subject to the linear constraints (1.2). We suggest the name LCOBYQA for the algorithm which solves linearly inequality constrained derivative free-optimization problems by quadraric models. The name LCOBYQA is an acronym for Linearly Constrained Opimination BY Quadratic Approximation. LCOBYQA algorithm is an extension of Powell's algorithms, (NEWUOA, and BOBYQA). The three algorithms are based on the main idea that each iteration constructs a full quadratic model from a few data, and use the remaining freedom in the model to minimize the Frobenius norm of the second derivative matrix of the change in the model. The three algorithms share many common features, but our algorithm differs from the other two algorithms in three main procedures, namely the initial calculation procedure, the trust-region subproblem minimization procedure and the geometry improvement procedure. LCOBYQA is a matlab software, which seeks the minimum of a function $f(\underline{x}),\ \underline{x} \in \mathbb{R}^n$, where the value of $f(\underline{x}),\ \underline{x} \in \mathbb{R}^n$ is specified by a subroutine provided by the user, which calculates the value of $f(\underline{x})$ for any given vector

$\underline{x} \in \mathbb{R}^n$, subject to linear constaints, but the derivatives of $f(\underline{x})$ are not availabl. The user of LCOBYQA has to provides a set of feasible interpolation points $\underline{x}_i,\ i = 1, 2, \ldots, m$, including a starting point $\underline{x}_0$, where $m = 2n + 1$. Also, the user provides the coefficient matrix $A^T \in \mathbb{R}^{\hat{m}\times n}$, the right hand side $\underline{b} \in \mathbb{R}^{\hat{m}}$ of the linear constraints, and the values of the parameters $\rho_{beg},\ \rho_{end}$. For each iteration k, the parameter ρ_k, which is a lower bound on the trust-region radius Δ_k, where $\Delta_k \geq \rho_k$, are also provided. The parameter ρ_{end}, which has to satisfy $\rho_{end} \leq \rho_{beg}$, should have the magnitude of the required accuracy in the final value of the variables. The initial interpolation points have the property $\| \underline{x}_i - \underline{x}_0 \|_\infty = \rho_{beg}$.

Let $\underline{x}_{opt}$, be an interpolation point such that $f(\underline{x}_{opt})$ is the least calculated value of $f(\underline{x})$ so far. The quadratic model at the k-th iteration is defined by

$$Q_k(\underline{x}_{opt} + \underline{p}_k) = f(\underline{x}_{opt}) + g_k^T \underline{p}_k + \frac{1}{2}\underline{p}_k^T G_k \underline{p}_k,\ \underline{p}_k \in \mathbb{R}^n, \tag{4.1}$$

its parameters being the vector $\underline{g}_k \in \mathbb{R}^n$ and the $n \times n$ symmetric matrix G_k. The model Q_k has to satisfy the interpolation conditions (3.52). Once the quadratic model is constructed, LCOBYQA is directed to one of the two iterations, namely the "trust-region" iteration or the "model" iteration. In each "trust-region" iteration, a step $\underline{p}_k$ from $\underline{x}_{opt}$, is defined as the vector that solves:

$$\begin{aligned} \min\ & Q_k(\underline{x}_{opt} + \underline{p}_k) = Q_k(\underline{x}_{opt}) + g_k^T \underline{p}_k + \frac{1}{2}\underline{p}_k^T G \underline{p}_k,\ \underline{p}_k \in \mathbb{R}^n, \\ \text{subject to:}\ & A^T(\underline{x}_{opt} + \underline{p}_k) \geq \underline{b},\ \ \| \underline{p}_k \| \leq \Delta_k. \end{aligned} \tag{4.2}$$

If it occurs that $\| \underline{p}_k \| < \frac{1}{2}\rho_k$, then $\underline{x}_{opt} + \underline{p}_k$ is considered to be sufficiently close to $\underline{x}_{opt}$, then the algorithm is switched to the "model" iteration. Usually, the inequality $\| \underline{p}_k \| \geq \frac{1}{2}\rho_k$ holds, and the new function value $f(\underline{x}_{opt} + \underline{p}_k)$ is calculated. Then we set $\rho_{k+1} = \rho_k$ and a new trust-region radius $\Delta_{k+1} \geq \Delta_k$ is chosen in the usual way that depends on the RATIO defined in (3.57) and the new point is defined by the equation (3.58). If RATIO< 0.1, the algorithm tests whether the geometry of the interpolation points needs to be improved or the trust-region to be shrinked. If the geometry of the interpolation points need to improve, then we switch to the " model iteration". The "model iteration" tries to improve the geometry of the interpolation points by choosing a

step $\underline{p}_k$ which maximizes the Lagrange polynomial $\ell_t(\underline{x}_{opt}+\underline{p}_k)$, subject to $A^T(\underline{x}_{opt}+\underline{p}_k) \geq \underline{b}$, $\| \underline{p}_k \| \leq \Delta_k$, where t is the index of the interolation point that must be removed in order to improve the model, as discussed in section 3.3, and each "model iteration" is followed by a "trust-region iteration". If RATIO ≥ 0.1, then the next iteration will be a "trust-region iteration", and so on the algorithm proceeds.

The structure of the remaining part of this chapter will be as follows. The selection of the initial interpolation points and the construction of the first quadratic model with formulae of initial matrix H and the factroization of the initial leading partition of H, are addressed in section 4.2. The updating of the model, when the new point is selected and the calling of RESCUE (3.6.3) procedure, are the subjects of section 4.3. The solution of the equation (4.2) using an active set version of truncated conjugate gradient method is described in section 4.4. In section 4.5, we address the manegement of the interpolation points. Further details of the algorithm are considered in section 4.6. A summary of the complete algorithm is given in section 4.7, with a few comments on its implementation. Finaly, the convergence of the algorithm is discussed in section 4.8.

4.2 Initial calculations

The user of LCOBYQA software has to supply the initial point $\underline{x}_0$, the coefficient matrix of the linear constraints A^T, the right hand side of the constraints $\underline{b}$, the parameters ρ_{beg}, ρ_{end}, an integer number $m = 2n + 1$, where n the number of variables, and the parameters ρ_1, Δ_1, which satisfy $\rho_1 = \Delta_1 = \rho_{beg}$. In order to construct the initial interpolation points, we need the point $\underline{x}_0$ to be strictly feasible, if $\underline{x}_0$ is not strictly feasible, the algorithm calls the phase one procedure of linear programming to provide a vertex point, and then uses the following steps to construct a strictly feasible point see (Igor et al., 2009).

Let $\underline{x}'$ be the vertex point which is generated by phase one procedure. Suppose that $\{1, 2, \ldots, l\}$ is the index set of the active constraints at $\underline{x}'$, i.e.,

$$\underline{a}_i^T \underline{x}' = b_i, \ i = 1, 2, \ldots, l. \tag{4.3}$$

For $\underline{d}$ to be a nonbinding feasible direction with respect to $i,\ i = 1, 2, \ldots, l$, it must satisfy

$$\underline{a}_i^T \underline{d} > 0. \tag{4.4}$$

Let $A_l = [a_1, \ldots, a_l]$, so $A_l^T \underline{d} > 0$, characterize the feasible direction $\underline{d}$ which is nonbinding with respect to $1, 2, \ldots, l$. Such direction can be found by solving the linear system

$$A_l^T \underline{d} = \underline{e}, \quad \underline{e} = [1, 1, \ldots, 1]^T. \tag{4.5}$$

The candidate point $\underline{x}_0$ is set to $\underline{x}' + \alpha \underline{d}, \quad 0 \leq \alpha \leq 2\Delta_1$. Initially, we set $\alpha = 2\Delta_1$, if $\underline{x}_0 = \underline{x}' + \alpha \underline{d}$ is strictly feasible, then the above steps, which we named procedure MOVE, are terminated. Otherwise, we reduce α iteratively. These steps produce a stirct feasible point. Once $\underline{x}_0$ is constructed, the first m interpolation points are given by:

$$\underline{x}_1 = \underline{x}_0, \quad \underline{x}_{i+1} = \underline{x}_0 + \rho_{beg} \underline{e}_i, \quad \underline{x}_{n+1+i} = \underline{x}_0 - \rho_{beg} \underline{e}_i, \quad i = 1, 2, \ldots, n, \tag{4.6}$$

where $\underline{e}_i$ is the i-th coordinate vector in $\mathbb{R}^n$. Consider the first quadratic model

$$Q_1(\underline{x}_0 + \underline{p}_1) = Q_1(\underline{x}_0) + \underline{p}_1^T \underline{g}_1(\underline{x}_0) + \frac{1}{2} \underline{p}_1^T G_1 \underline{p}_1, \quad \underline{p}_1 \in \mathbb{R}^n, \quad \text{subject to} \quad A^T(\underline{x}_0 + \underline{p}_1) \geq \underline{b}, \tag{4.7}$$

that satisfies the interpolation conditions

$$Q_1(\underline{x}_i) = f(\underline{x}_i), \quad i = 2, \ldots, m. \tag{4.8}$$

The vecter $\underline{g}_1$ and the diagonal elements $(G_1)_{ii},\ i = 1, 2, \ldots, n$, are given uniquely by the conditions (4.8). The initial calulations of LCOBYQA also set the initial matrix $H = W^{-1}$ of the first model, where W occurs in the linear system (3.11).

It is staightforward to derive the element of W for the initial points $\underline{x}_i, \quad i = 1, 2, \ldots, m$, but it is required to have the elements of Ξ and Υ explicitly, with the factorization of Ω (see equation 3.53). Fortunately, the chosen positions of the initial interpolation points provide convenient formulae for all of these terms see (Powell, 2006). The first row of the initial $(n + 1) \times m$ matrix Ξ has the simple form

$$\Xi_{1j} = \delta_{1j}, \quad j = 1, 2, \ldots, m. \tag{4.9}$$

Further, for integer $i = 2, 3, \ldots, n+1$, the i-th row of the initial Ξ also has just two nonzero elements

$$\Xi_{ii} = (2\rho_{beg})^{-1} \quad \text{and} \quad \Xi_{ii+n} = -(2\rho_{beg})^{-1}. \tag{4.10}$$

This completes the definition of Ξ for the initial interpolation points. Moreover, the initial $(n+1) \times (n+1)$ matrix Υ, is identically zero. The factorization of Ω, which guarantees that the rank of Ω is at most m-n-1, having the form

$$\Omega = \sum_{k=1}^{m-n-1} \mathsf{v}_k \mathsf{v}_k^T = VV^T, \tag{4.11}$$

where the components of the initial vector $\mathsf{v}_k \in \mathbb{R}^m$, which is the k-th column of V, are given the values

$$V_{1k} = -\sqrt{2}\rho_{beg}^{-2}, \qquad V_{k+1k} = \frac{1}{2}\sqrt{2}\rho_{beg}^{-2},$$

$$V_{k+n+1k} = \frac{1}{2}\sqrt{2}\rho_{beg}^{-2}, \qquad V_{jk} = 0, \quad \text{otherwise, where } 1 \leq k \leq n,\ j = 1, 2, \ldots, m. \tag{4.12}$$

We see that each of these columns has just three nonzero elements.

Let $\underline{x}_{opt}$ be an interpolation point such that $f(\underline{x}_{opt})$ is the least calculated value of f so far. As mentioned in section one of this chapter, each trust-region iteration solves a subprobem of the form (4.2), using a version of the active set method for indefinite quadratic programming. However, this method requires that the initial reduced Hessiam matrix at $\underline{x}_{opt}$ must be positve definite. Therefore, if the reduced Hessian at $\underline{x}_{opt}$ is not positive definite, artificial constraints are added by the algorithm to the initial working set (the set of active constraints at $\underline{x}_{opt}$). These constraints involve artificial variables y_i, and are of the form $y_i \geq (\underline{x}_{opt})_i$ or $y_i \leq (\underline{x}_{opt})_i$. The purpose of the artificial constraints is to convert the reduced Hessian matrix at $\underline{x}_{opt}$ to a positive definite matrix (Gill et al., 1988). When the iterations of the algorithm proceed, the artificial constraints are removed automatically. The description of the initial calculation is thus complete.

4.3 Updating procedure

In this section we breifly discuss the changes that are made to the quadratic model Q, and to the inverse matrix H of equation (3.41), when a new interpolation point replaces

an old one. For more detail see (Powell, 2004b). Also, we discuss the condition under which the procedure RESCUE should be called (for a detailed discription see section 3.6). Let t be the index of the interpolation point, that will be removed. Thus a new point, say $\underline{x}^+$, should be added. Our new interpolation points will be:

$$\underline{x}^+ = \underline{x}_t^+ = \underline{x}_{opt} + \underline{p}_k,$$

$$\underline{x}_i^+ = \underline{x}_i, \quad i \in \{1, 2, \ldots, m\} \setminus \{t\}. \tag{4.13}$$

As mentioned in section 3.4, the new inverse matrix is given by

$$H^+ = H + \frac{1}{\sigma_t}\left[\alpha_t(\underline{e}_t - H\underline{w})(\underline{e}_t - H\underline{w})^T - \beta_t H\underline{e}_t\underline{e}_t^T H\right] +$$

$$\frac{1}{\sigma_t}\left[\tau_t\left\{H\underline{e}_t(\underline{e}_t - H\underline{w})^T + (\underline{e}_t - H\underline{w})\underline{e}_t^T H\right\}\right],$$

where the parameters of H^+ are:

$$\alpha_t = \underline{e}_t^T H\underline{e}_t, \qquad \beta_t = (\underline{e}_t - H\underline{w})^T\underline{w} + \eta_t,$$

$$\tau_t = \underline{e}_t^T H\underline{w}, \qquad \sigma_t = \alpha_t\beta_t + \tau_t^2,$$

and $\underline{w}$ is the vector defined in equation (3.44). Once H^+ is constructed, the submatrices Ξ and Υ of equation (3.53) are overwritten by Ξ^+ and Υ^+ respectively, and the facorization of Ω^+ is stored instead of Ω. The purpose of the factorization is to reduce the damage from rounding errors to the identity $W = H^{-1}$, which is fullfilled at the beginning of each iteration. It has become obvious from numerical exprements (Powell, 2006), that huge rounding errors may occur in H in practice, including a few negative values of $H_{ii},\ 1 \le i \le m$. Also, Ω should be positive semi-definite. Therefore, we consider the process of updating H when H is very different from W^{-1}, assuming that the calculation of the current iteration is exact. Then H^+ is the inverse of the matrix that has the elements

$$W_{it}^+ = W_{ti}^+ = (W^+\underline{e}_t)_i, \quad i = 1, 2, \ldots, m+n+1,$$

$$W_{ij}^+ = W_{ij} = H_{ij}^{-1}, \quad \text{otherwise}, \quad 1 \le i, j \le m+n+1, \tag{4.14}$$

which gives the identity

$$(H^+)_{it}^{-1} = W_{it}^+ \quad \text{and} \quad (H^+)_{ti}^{-1} = W_{ti}^+, \quad i = 1, 2, \ldots, m+n+1,$$

$$W_{ij}^{+}-(H^{+})_{ij}^{-1}=W_{ij}-H_{ij}^{-1}\ \text{ otherwise},\quad 1\leq i,j\leq m+n+1. \tag{4.15}$$

In other words, overwritting of W and H by W^{+} and H^{+} makes no difference to the elements of $W-H^{-1}$, except that the t-th row and column of this error matrix becomes zero. It follows that when all the current interpolation points have been discarded by future iterations, then all the current errors in the first m rows and columns of $W-H^{-1}$ will be annihlated. Equation (4.15) suggests that any errors in the bottom right $(n+1)\times(n+1)$ submatrix H^{-1} are retained.

The factorization of Ω provides a perfect remedy of this situation. Indeed, if H is any nonsingular $(m+n+1)\times(m+n+1)$ matrix, and if the rank of the leading $m\times m$ submatrix Ω is $m-n-1$, then the bottom right $(n+1)\times(n+1)$ submatrix H^{-1} is zero (the proof is found in (Powell, 2004b)). Thus, the very welcoming property $(H^{+})_{ij}^{-1}-W^{+}=0,\quad m+1\leq i,j\leq m+n+1$ is guaranteed by the factorization $\Omega^{+}=V^{+}V^{+T}$. Let the factorization $\Omega=VV^{T}$ be given, the new factorization can be constructed by changing only one column of V, the first column say, see (Powell, 2009). Specifically, the first column has the form

$$V_{i1}^{+}=\sigma_{t}^{-\frac{1}{2}}\left[\tau_{t}V_{i1}+(\underline{e}_{t}-\underline{e}_{opt}-H\left\{\underline{w}-\underline{v}\right\})_{i}V_{t1}\right],\ i=1,2,\ldots,m, \tag{4.16}$$

where $\underline{v}\in\mathbb{R}^{m+n+1}$ is defined by

$$\begin{aligned}v_{i}&=\frac{1}{2}\left\{(\underline{x}_{i}-\underline{x}_{0})^{T}(\underline{x}_{opt}-\underline{x}_{0})\right\}^{2},\quad i=1,2,\ldots,m,\\ v_{m+1}&=1,\quad v_{m+1+i}=(\underline{x}_{opt}-\underline{x}_{0})_{i},\quad i=1,2,\ldots,n.\end{aligned} \tag{4.17}$$

We observed that $\underline{v}$ is the opt-th column of the matrix W, so from equation 3.11 we have $H\underline{v}=\underline{e}_{opt}$.

Another feature of the storage and updating of H by LCOBYQA is that: when $\underline{p}_{k}$ in the subproblem (4.2) is constructed, the constant term of Q is irrelevent. Moreover, the constant term of Q_{old} is also not required, because the identities $Q_{old}(\underline{x}_{opt})=f(\underline{x}_{opt})$ and $\underline{x}^{+}=\underline{x}_{opt}+\underline{p}_{k}$, allow the parameters of the system (3.11) to be written in the form

$$\left(\begin{array}{c}\underline{\lambda}\\ \hline c\\ \\ \underline{g}\end{array}\right)=\left\{(f(\underline{x}_{opt}+\underline{p}_{k})-f(\underline{x}_{opt}))-(Q_{old}(\underline{x}_{opt}+\underline{p}_{k})-Q_{old}(\underline{x}_{opt}))\right\}H^{+}\underline{e}_{t}. \tag{4.18}$$

Therefore, LCOBYQA does not store the constant term of any quadratic model. It follows that c^+ in equation (3.2) is ignored, which makes the $(m+1)$-th row and column of H^+ unnecessary for the revision of Q. That is equivalent to the process of removing the first row of every submatrix Ξ and the first row and column of every submatrix Υ, but the other elements are retained. The following procedure is used by LCOBYQA to update H without its $(m+1)$-th row and column.

From equations (3.44) and (4.17), the terms $H\underline{w}$ and $\underline{w}^T H\underline{w}$ take the values

$$H\underline{w} = H(\underline{w} - \underline{v}) + \underline{e}_{opt}, \tag{4.19}$$

$$\underline{w}^T H\underline{w} = (\underline{w} - \underline{v})^T H(\underline{w} - \underline{v}) + 2\underline{w}^T \underline{e}_{opt} - \underline{v}^T \underline{e}_{opt}. \tag{4.20}$$

These formulae allow the parameters of H^+ to be calculated without the $(m+1)$-th row and column of H, because the $(m+1)$-th component of $\underline{w} - \underline{v}$ is zero. Once H^+ is constructed, the parameter g^+ is given from equation (4.18), and G^+ is calculated using equations (3.37), (3.38) and (3.39).

Finally, we address breifly the calling of RESCUE procedure. LCOBYQA tests the condition (3.64) at each iteration. If this condition is satisfied, then the algorithm LCOBYQA calls RESCUE procedure. Detailed desciption of RESCUE is found in section 3.6.3.

4.4 The trust-region subproblem procedure

The trust-region subproblem is discussed in detail in this section. Consider the problem

$$\min \; Q_k(\underline{x}_{opt} + \underline{p}_k) = Q_k(\underline{x}_{opt}) + \underline{g}_k^T \underline{p}_k + \frac{1}{2}\underline{p}_k^T G_k \underline{p}_k, \; \underline{p}_k \in \mathbb{R}^n,$$

$$\text{subject to: } \; A^T(\underline{x}_{opt} + \underline{p}_k) \geq \underline{b}, \quad \| \underline{p}_k \| \leq \Delta_k,$$

where the parameters $\underline{g}_k$, G_k are given data. We will use a null space active set version of the truncated conjugate gradient procedure (for indefinite quadratic programming) to solve the above subproblem. For more details see ((Byrd et al., 2002), (Gay and David, 1984), (Gill et al., 1991), (Zhibin and Binliang, 2006), (Colson et al., 2004), (Colson et al., 2003), (Dong and Shaoman, 2002), (Ju et al., 2004), (Vadi and Avi, 1985), (XU et al.,

2004), (Ying-Ji et al., 2007), and (More and Sorensen, 1983)). The idea of choosing the active set method in this work, is motivated by the fact that, if the correct working set at the solution is known a priori, then the solution of the linear equality constrained problem would also be a solution to a linear inequality problem.

Assume that s constraints are active at $\underline{x}_{opt}$, let $\hat{A}^T$ denote the matrix whose rows correspond to the active constraits at $\underline{x}_{opt}$, and $\hat{\underline{b}}$ be the corresponding right hand side vector. Therefore, in order to solve (4.2), we solve the subproblem

$$\min \; Q_k(\underline{x}_{opt}+\underline{p}_k) = Q_k(\underline{x}_{opt}) + \underline{g}_k^T\underline{p}_k + \frac{1}{2}\underline{p}_k^T G_k \underline{p}_k, \; \underline{p}_k \in \mathbb{R}^n,$$
$$\text{subject to:} \;\; \hat{A}^T(\underline{x}_{opt}+\underline{p}_k) = \hat{\underline{b}}, \;\; \| \underline{p}_k \| \leq \Delta_k, \tag{4.21}$$

where $\hat{A}^T \in \mathbb{R}^{s\times n}$, $\hat{\underline{b}} \in \mathbb{R}^s$ and $\hat{A}^T$ is full rank. Let Î be the index set of active constraints at $\underline{x}_{opt}$ (the working set at $\underline{x}_{opt}$). As mentioned in section 2.3.1, any step $\underline{p}_k$ from a feasible point to any other feasible point must satisfy:

$$\hat{A}^T\underline{p}_k = \underline{0}. \tag{4.22}$$

Now, let Y and Z be an $n\times s$ and $n\times(n-s)$ matrices, respectively, such that $[Y:Z]$ is nonsingular. In addition, let $\hat{A}^TY = I$ and $\hat{A}^TZ = O$, the columns of Z form a basis for the null space of $\hat{A}^T$. So from equation (4.22), any feasible direction $\underline{p}_k$ can be written as $\underline{p}_k = Z\underline{y}$, where $\underline{y}$ is any vector in $\mathbb{R}^{n-s}$. Therefore, any solution of $\hat{A}^T\underline{x} = \hat{\underline{b}}$ is given by $\underline{p}_k = Y\hat{\underline{b}} + Z\underline{y}$. Thus, (4.21) could be written as:

$$\min_{\underline{y}\in\mathbb{R}^{n-s}} \; \psi(\underline{y}) = \frac{1}{2}\underline{y}^TZ^TGZ\underline{y} + (g+GY\hat{\underline{b}})^TZ\underline{y} + \frac{1}{2}(g+GY\hat{\underline{b}})^TY\hat{\underline{b}}, \quad \text{subject to:} \;\; \| \underline{y} \| \leq \Delta_r, \tag{4.23}$$

where $\Delta_r = \sqrt{\Delta_k^2 - \| Y\hat{\underline{b}} \|^2}$. See (Frank and Bersini, 2005). We observe that the constant term of equation (4.23) is independent of $\underline{y}$, so that we can rewrite this problem in the form

$$\min_{\underline{y}\in\mathbb{R}^{n-s}} \; \psi(\underline{y}) = \frac{1}{2}\underline{y}^TZ^TGZ\underline{y} + (g+GY\hat{\underline{b}})^TZ\underline{y}, \quad \text{subject to:} \;\; \| \underline{y} \| \leq \Delta_r. \tag{4.24}$$

Once the solution $\underline{y}^\star$ is calculated, then we substitute

$$\underline{p}_k = Y\hat{\underline{b}} + Z\underline{y}^\star. \tag{4.25}$$

As mentioned at the begining of this section, we used an active set version of the truncated conjugate gradient method to find $\underline{y}^{\star}$ see (Conn et al., 2000). This method produces a piecewise linear path in $\mathbb{R}^{n-s}$, begining at the centre $\underline{y}_0 = \underline{0}$ of the trust-region $\left\{\underline{y} : \| \underline{y} \| \leq \Delta_r\right\}$. For $j \geq 1$, $\underline{y}_j$ is then generated by:

$$\underline{y}_j = \underline{y}_{j-1} + \alpha \underline{s}_j, \tag{4.26}$$

where $\underline{s}_j$ is the direction of $\underline{y}_j$ which is given by

$$\underline{s}_j = \begin{cases} -Z^T \nabla Q_k(\underline{x}_{opt}) \ , & j = 1, \\ -Z^T \nabla Q_k(\underline{x}_{opt} + Z\underline{y}_{j-1}) + \beta_j \underline{s}_{j-1}, & j \geq 2, \end{cases} \tag{4.27}$$

and $\beta_j = \| Z^T \nabla Q_k(\underline{x}_{opt} + Z\underline{y}_{j-1}) \|^2 \ / \ \| Z^T \nabla Q_k(\underline{x}_{opt} + Z\underline{y}_{j-2}) \|^2$.

For each iteration of the subproblem (4.24), let α_Δ be the step along the direction $\underline{s}_j$ to the trust-region boundary. Let α_ψ be such that the derivative of $\psi(\underline{y}_{j-1} + \alpha_\psi \underline{s}_j)$ with respect to α_ψ is zero, except that α_ψ is regarded as infinity if the first derivative with respect to α_ψ is negative for every positive α_ψ. Let α_c be such that

$$\alpha_c = \min_{i \notin \hat{I}} \left(\frac{\hat{b}_i - \underline{a}_i^T (Y\underline{\hat{b}} + Z\underline{y}_j)}{\underline{a}_i^T (Y\underline{\hat{b}} + Z\underline{y}_j)}, \quad \underline{a}_i^T (Y\underline{\hat{b}} + Z\underline{y}_j) < 0 \right) \tag{4.28}$$

These $\alpha' s$ are calculated, the chosen step length α being the least of them, and $\underline{y}_{j-1}$ is overwritten by $\underline{y}_{j-1} + \alpha \underline{s}_j$. In the case $\alpha = \alpha_\Delta$, the trust-region boundary has been reached which completes the iteration of the conjugate gradient method. If $\alpha = \alpha_c$, the current line search is restricted by a constraint. Its index is added to $\hat{I}$ so that the subsequent choice of $\underline{x}_{opt} + (Y\underline{\hat{b}} + Z\underline{y}_j)$ will remain on the boundary of the additional constraint. At this stage $Q(\underline{x}_{opt}) - Q(\underline{x}_{opt} + (Y\underline{\hat{b}} + Z\underline{y}_j))$ is the total reduction in Q that occurred so far, and the product $\| \nabla Q(\underline{x}_{opt} + (Y\underline{\hat{b}} + Z\underline{y}_j)) \| \Delta_k$ is likly to be an upper bound on any further reduction. Therefore, the termination occurs if the condition

$$\| \nabla Q(\underline{x}_{opt} + (Y\underline{\hat{b}} + Z\underline{y}_j)) \| \Delta_k \leq 0.01 \left\{ Q(\underline{x}_{opt}) - Q(\underline{x}_{opt} + (Y\underline{\hat{b}} + Z\underline{y}_j)) \right\} \tag{4.29}$$

is achieved. Otherwise, the conjugate gradient procedure is restarted at the point $\underline{x}_{opt} + (Y\underline{\hat{b}} + Z\underline{y}_j)$ with $\underline{s}_j = -Z^T \nabla Q(\underline{x}_{opt} + (Y\underline{\hat{b}} + Z\underline{y}_j))$ as the next search direction. In the

remaining case $\alpha \leq \alpha_{\Delta},\ \alpha \leq \alpha_c,\ \alpha = \alpha_{\psi}$, α is a full projected conjugate gradient step without any interference from any constraint which gives the strict reduction in Q_k. If this reduction is at most the right hand side of (4.29) or the inequality (4.29) holds at the new point $\underline{x}_{opt} + (Y\hat{\underline{b}} + Z\underline{y}_j)$, then the termination also occurs. The alternative is a line search from the new point along the direction $\underline{s}_j$, which is chosen as a projected steepest desent direction $-Z^T\nabla Q(\underline{x}_{opt} + (Y\hat{\underline{b}} + Z\underline{y}_j))$ augmented by the multiple of the previous search direction that gives the orthogonality to the change ∇Q that occurred in previous iteration. If $\underline{x}_{opt} + \underline{p}_k$ minimizes $f(\underline{x})$ with respect to the consraints in $\hat{I}$, then we calculate the Lagrange multipliers at $\underline{x}_{opt} + \underline{p}_k$. If the Lagrange multipliers are nonnegative, then the agorithm is terminated, otherwise we delete the constraint which corresponds to the least Lagrange multiplier from $\hat{I}$. The solution of the trust-region subproblem is thus complete.

4.5 Geometry improvement of the interpolation points procedure

As mentioned in the introduction of this chapter, if RATIO of equation (3.57) satisfies RATIO < 0.1, then the algorithm tests whether the geometry of the interpolation points needs to be improved. In this section, we will discuss in detail the improvement of the interpolation points using the Lagrange polynomials mentioned in section 3.3.
If RATIO < 0.1, we set t to be an integer in $[1, m]$ such that $\underline{x}_t$ maximizes the distance

$$\text{DIST} = \| \underline{x}_i - \underline{x}_{opt} \|, \quad i = 1, 2, \ldots, m. \tag{4.30}$$

If DIST $\geq 2\Delta_k$, then the procedure that improves the geometry of the interpolation points is to be invoked. This procedure replaces the current interpolation point $\underline{x}_t$ by a new point $\underline{x}^+$, in order to improve the geometry of the interpolation points. This is done by using the Lagrange interpolation polynomials. As mentioned in section 3.3, the Lagrange interpolation polynomial is a quadratic polynomial $\ell_t(\underline{x}),\ \underline{x} \in \mathbb{R}^n$, that satisfies the Lagrange conditions of equation (3.27) and the remaining degrees of freedom are used

to minimizing the Frobenius norm $\| \nabla^2 \ell_t(\underline{x}) \|_F$. Therefore, ℓ_t is the quadratic function

$$\ell_t(\underline{x}) = c + (\underline{x} - \underline{x}_0)^T \underline{g} + \frac{1}{2} \sum_{k=1}^{m} \lambda_k \left\{ (\underline{x} - \underline{x}_0)^T (\underline{x}_k - \underline{x}_0) \right\}^2, \quad \underline{x} \in \mathbb{R}^n. \tag{4.31}$$

The parameters $c,\ g$ and $\lambda_k,\ k = 1, 2, \ldots,\ m$, being defined by the linear system (3.11), where the right hand side is the coodinate vector $\underline{e}_t \in \mathbb{R}^{m+n+1}$. Thus, the parameters of ℓ_t are the t-th column of the matrix H. As mentioned in equation (3.51) of section 4.3 that $\tau_t = \underline{e}_t H \underline{w} = \ell_t(\underline{x}_{opt} + \underline{p}_k)$, we expect relatively larger modulus of the denominater $\sigma_t = \alpha_t \beta_t + \tau_t^2$ to be beneficial when the formula (3.51) is applied. Therefore, when the geometry of the interpolation points need to be improved, the point $\underline{x}_t$ is replaced by the point $\underline{x}^+ = \underline{x}_{opt} + \underline{p}_k$, where the direction $\underline{p}_k$ solves the following equation

$$\max \ \ | \ \ell_t(\underline{x}_{opt} + \underline{p}_k) \ |, \ \ \text{subject to} \ \ \| \ \underline{p}_k \ \| \leq \Delta_k, \ \ \hat{A}^T (\underline{x}_{opt} + \underline{p}_k) \geq \hat{\underline{b}}. \tag{4.32}$$

It is reported in (Powell, 2008, 2009) that $\underline{p}_k$ is selected usually from one of $m-1$ straight lines in $\mathbb{R}^n$ through $\underline{x}_{opt}$ and other interpolation points. Let $\underline{x}_t$ be the point that will be removed in order to improve the interpolation set, then the direction $\underline{p}_k$ is chosen as $\underline{p}_k = \alpha(\underline{x}_t - \underline{x}_{opt})$, where $\alpha \in (0, 1)$ such that $| \ \ell_t(\underline{x} + \underline{p}_k) \ |$ is maximum. Clearly, $| \ \ell_t(\underline{x} + \underline{p}_k) \ |$ is positive, since $\ell_t(\underline{x}_t) = 1$. The above choice of $\underline{p}_k$ guarantees that $\underline{x}^+ = \underline{x}_{opt} + \underline{p}_k$ is feasible, since the straight line between any two feasible points of a convex set (linear constraints) is feasible. However, there is a fundamental disadvantage, in choosing $\underline{p}_k$ along the direction $(\underline{x}_t - \underline{x}_{opt})$. It is that, if the steps $\underline{x}_j - \underline{x}_{opt},\ j = 1, 2, \ldots, m$, fail to span $\mathbb{R}^n$. For then this property is inherited by the new steps $\underline{x}_j^+ - \underline{x}_{opt},\ j = 1, 2, \ldots,\ m$. The description of the improvement procedure is thus complete.

4.6 Further details of LCOBYQA

In this section, we address some issues that are presented in (Powell, 2006, 2009) and we are not discussed in previos sections. They involve the selection of the point $\underline{x}_t$ that will replaced the point $\underline{x}^+ = \underline{x}_{opt} + \underline{p}_k$, where $\underline{p}_k$ is the solution of the subproblem (4.2). We also discuss the revising of Δ_k, and ρ_k at each iteration, and we also discuss

the conditions that are used to test whether three values of $\mid f - Q \mid$ and $\underline{p}_k$ are small, which is appeared in step 3 of our algorithm. Finaly, we address the shift of the origin $\underline{x}_0$. First we discuss the selection of the point $\underline{x}_t$. The selection of $t \in [1, 2, \ldots, m]$ provides a relatively large denominater $\mid \sigma_t \mid = \mid \alpha_t \beta_t + \tau_t^2 \mid$. Specifically, t is set to the integer in the set $\{1, 2, \ldots, m\} \setminus \{opt\}$ that maximizes the weighted denominater

$$\max \ \left[1, \| \underline{x}_t - \underline{x}_{opt} \|^2 / \Delta_k^2\right] \left\{ H_{tt}(\frac{1}{2} \| \underline{x}^+ - \underline{x}_0 \|^4 - \underline{w}^T H \underline{w}) + (\underline{e}_t^T H \underline{w})^2 \right\}. \quad (4.33)$$

For justification of this choice, see (Powell, 2009).

The choice of Δ_{k+1} in the "trust-region" iteration that calculates $f(\underline{x}_{opt} + \underline{p})$, depends on $\underline{p}_k$ and RATIO (3.57). Specifically, we use the formula

$$\Delta_{k+1} = \begin{cases} \min[\frac{1}{2}\Delta_k, \| \underline{p}_k \|], & \text{RATIO} \leq 0.1 \\ \max[\frac{1}{2}\Delta_k, \| \underline{p}_k \|], & 0.1 < \text{RATIO} \leq 0.7 \\ \max[\frac{1}{2}\Delta_k, 2 \| \underline{p}_k \|], & \text{RATIO} > 0.7, \end{cases} \quad (4.34)$$

except that Δ_{k+1} is set to the ρ_k if the value of (4.34) is at most $1.5\rho_k$,

If the value of ρ_k is decreased from ρ_k to ρ_{k+1}, the reduction is by a factor of 10, unless only one or two changes are going to attain the final value of ρ_{end}. LCOBYQA applies the following formula to revise ρ_k

$$\rho_{k+1} = \begin{cases} \rho_{end}, & \rho_k \leq 16\rho_{end} \\ (\rho_k \rho_{end})^{\frac{1}{2}}, & 16\rho_{end} < \rho_k \leq 250\rho_{end} \\ 0.1\rho_k, & \rho_k > 250\rho_{end}. \end{cases} \quad (4.35)$$

As in Step 3 of algorithm 4.1 presented in section 4.7, if the step $\underline{p}_k$ has the property $\| \underline{p}_k \| < \frac{1}{2}\rho_k$, we test whether three recent values of $\mid f - Q \mid$ and $\| \underline{p}_k \|$ are small. Let CRVMIN be the least eigenvalue of $\nabla^2 Q$, we use the value of CRVMIN in this test. We prefer not to calculate $f(\underline{x}_{opt} + \underline{p}_k)$ when the predicted reduction in f, namely $Q(\underline{x}_{opt}) - Q(\underline{x}_{opt} + \underline{p}_k)$ is less than $\frac{1}{8}$CRVMIN. Further, if the value of the error $f(\underline{x}_{opt} + \underline{p}_k) - Q(\underline{x}_{opt} + \underline{p}_k)$ on recent three iterations are also less than $\frac{1}{8}$CRVMIN, then we take the view that trying to improve the accuracy of the model would be a waste of time.

Specifically, if all the conditions $\| \underline{p}_k \| < \frac{1}{2}\rho_k$ and $| f(\underline{x}_{opt} + \underline{p}_k) - Q(\underline{x}_{opt} + \underline{p}_k) | \leq \frac{1}{8}$CRVMIN hold for three recent values, then the algorithm reduces ρ_k by a factor of 10 if $\rho_k \geq \rho_{end}$, and also reduces Δ_k to $\max[\frac{1}{2}\rho_k, \rho_{k+1}]$.

The updating of the origin $\underline{x}_0$ is very important for reducing rouding errors. Numerical expremments (Powell, 2006) using difficult objective functions, after a sequence of many iterations detect that there is an unacceptable errors if

$$\| \underline{x}_{opt} - \underline{x}_0 \| \geq 10^{2.5} \| \underline{p}_k \| . \tag{4.36}$$

Therefore, LCOBYQA tests the condition

$$\| \underline{p}_k \|^2 \leq 10^{-3} \| \underline{x}_{opt} - \underline{x}_0 \|^2 \tag{4.37}$$

before replacing $\underline{x}_t$ by $\underline{x}_{opt} + \underline{p}_k$ in the k-th iteration. If condition (4.37) holds, then $\underline{x}_0$ is overwritten by $\underline{x}_{opt}$ which alters the last n rows of X of equation (3.13) and all elements of (3.12). Details of this task is cosidered in (Powell, 2004a), only a brief outline is given below to the changes that are made to H when $\underline{x}_0$ is shifted. Let $\underline{x}_{av}$ and $\underline{r}$ be the vectors $\frac{1}{2}(\underline{x}_0 + \underline{x}_{opt})$ and $(\underline{x}_{opt} - \underline{x}_0)$, respectively, before $\underline{x}_0$ is overwritten by $\underline{x}_{opt}$, let χ be the $n \times m$ matrix that has the columns

$$\underline{y}_j = \left\{ \underline{r}^T (\underline{x}_j - \underline{x}_{av}) \right\} (\underline{x}_j - \underline{x}_{av}) + \frac{1}{4} \| \underline{r} \|^2 \underline{r}, \ j = 1, 2, \ldots, m, \tag{4.38}$$

and let Θ_{old} and Θ_{new} be the old and new matrices H and H^+ without their $(m+1)$-th rows and columns. Then according to equations (5.11) and (5.12) of (Powell, 2004a), Θ_{new} is defined by

$$\Theta_{new} = \left(\begin{array}{c|c} I & O \\ \hline \chi & I \end{array} \right) \Theta_{old} \left(\begin{array}{c|c} I & \chi^T \\ \hline O & I \end{array} \right) \tag{4.39}$$

Thus, the product $\chi\Omega$ and the sum $\chi\Xi_{Red}^T + \Xi_{Red}\chi^T + \chi\Omega\chi^T$ are added to the last n rows of Ξ and to the trailing $n \times n$ submatrix of Υ, respectively, where Ξ_{Red} is the original matrix Ξ without its first row. When $\underline{x}_0$ is overwritten by $\underline{x}_{opt}$, the gradient $\nabla Q(\underline{x}_0)$ has to be revised too. The new gradient is given by

$$\nabla Q(\underline{x}_{opt}) = \nabla Q(\underline{x}_0) + \nabla^2 Q . \underline{r}, \tag{4.40}$$

where $\underline{r} = \underline{x}_{opt} - \underline{x}_0$.

Also $\nabla^2 Q$ is modified according to

$$\nabla^2 Q = \Gamma + \sum_{j=1}^{m} \gamma_j (\underline{x}_j - \underline{x}_0)(\underline{x}_j - \underline{x}_0)^T = \Gamma + \sum_{j=1}^{m} \gamma_j (\underline{x}_j - \underline{x}_{opt} + \underline{r})(\underline{x}_j - \underline{x}_{opt} + \underline{r})^T =$$

$$\Gamma + \underline{\upsilon r}^T + \underline{r\upsilon}^T + \sum_{j=1}^{m} \gamma_j (\underline{x}_j - \underline{x}_{opt})(\underline{x}_j - \underline{x}_{opt})^T, \tag{4.41}$$

where

$$\underline{\upsilon} = \sum_{j=1}^{m} \gamma_j (\underline{x}_j - \underline{x}_{opt} + \frac{1}{2}\underline{r}) = \sum_{j=1}^{m} \gamma_j (\underline{x}_j - \underline{x}_{av}) \tag{4.42}$$

Therefore, the shift in $\underline{x}_0$ requires $\underline{\upsilon r}^T + \underline{r\upsilon}^T$ to be added to Γ, although the parameters $\gamma_j,\ j = 1, 2, \ldots, m$, are unchanged.

4.7 Summery of LCOBYQA algorithm

We now introduce the summary of our algorithm which solves unconstrained, simple bound constrained and linear constrained derivative free optimization problems. The following summary of the algorithm is divided into nine steps, where each step refers to the relevant part of the material of the previous sections.

Algorithm 4.1 [LCOBYQA Algorithm]:

- **Step 1:**

  The user of the algorithm supplies the following data: The initial point $\underline{x}_0$,
  the parameters ρ_{beg}, ρ_{end}, the coefficient matrix of the the constraints A^T, and the right hand side vector $\underline{b}$

- **Step 2:** Initialization

  Set $m = 2n + 1$, where n the number of variables.
  If the initial point $\underline{x}_0$ is not strictly feasible, invoke the procedures PHASE ONE and MOVE to a generate strictly feasible point.

Set $\Delta = \rho_{beg},\ \rho_1 = \rho_{beg}$.

Construct the initial interpolation points.

Select a point $\underline{x}_{opt}$ such that $f(\underline{x}_{opt})$ has least value of $f(\underline{x})$ so far.

Select $\hat{I}$ to be the initial working set at $\underline{x}_{opt}$, set $\hat{A}^T$ to be the coefficient martix of the constraints in the working set, set $\hat{\underline{b}}$ to be the corresponding right hand side vector.

Construct the first quadratic model Q, set $k = 1$, set $Ar = 0$.

- **Step 3:**

if $\rho_k \leq \rho_{end}$

go to step 9

end(if).

Calculate the step $\underline{p}_k$ that minimize problem (4.21).

If $\| \underline{p}_k \| < \frac{1}{2}\rho_k$

set $Ar = 1$

if three recent values of $| f(\underline{x}_{opt}) - Q(\underline{x}_{opt}) |$ and $\| \underline{p}_k \|$ are small

go to step 8

else

reduce Δ, set RATIO=-1, go to step 6

end(if)

end(if)

- **Step 4:**

Calculate $f(\underline{x}_{opt} + \underline{p}_k)$, and RATIO (3.57).

Revise Δ, subject to $\Delta \geq \rho_k$.

If RATIO $\geq .1$

Select an integer t (from 4.33), the index of the interpolation point that will be dropped

else

set $t = 0$

end(if).

- **Step 5:**

  If $t > 0$

  Update the model Q, such that Q interpolate $f(\underline{x})$ at $\underline{x}_{opt} + \underline{p}_k$ instead of $\underline{x}_t$.

  If $\underline{x}_{opt} + \underline{p}_k$ minimize $f(\underline{x})$ with respect to the consraints in $\hat{I}$

  if the Lagrange multipliers at $\underline{x}_{opt} + \underline{p}_k$ are nonnegative

  go to step 9

  else

  delete a constraint with least Lagrange multiplier from the working set,

  update $\hat{I}$, $\hat{A}^T$, $\hat{\underline{b}}$

  end(if)

  end(if)

  If $\underline{x}_{opt} + \underline{p}_k$ reaches the boundary of a constraint not in the working set

  add this constraint to the working set, update $\hat{I}$, $\hat{A}^T$, $\hat{\underline{b}}$

  end(if)

  If $f(\underline{x}_{opt} + \underline{p}_k) < f(\underline{x}_{opt})$

  overwrite $\underline{x}_{opt}$ by $\underline{x}_{opt} + \underline{p}_k$

  end(if)

  end(if)

  If RATIO≥ 0.1

  set $k = k + 1$, go to step 3

  end(if)

- **Step 6:**

  Select an integer t that maximizes the distance DIST $= \| \underline{x}_t - \underline{x}_{opt} \|$

  If DIST$\geq 2\Delta$

  Replace $\underline{x}_t$ by $\underline{x}_{opt} + \underline{p}_k$, where solves problem (4.32)

  Update the model Q, such that Q interpolate $f(\underline{x})$ at $\underline{x}_{opt} + \underline{p}_k$ instead of $\underline{x}_t$.

  If $f(\underline{x}_{opt} + \underline{p}_k) < f(\underline{x}_{opt})$

  overwrite $\underline{x}_{opt}$ by $\underline{x}_{opt} + \underline{p}_k$.

end(if)

set $k = k + 1$, go to step 3

end(if).

- **Step 7:**

If $\max\ [\| \underline{p}_k \|, \Delta] > \rho_k$ or RATIO> 0

set $k = k + 1$, go to step 3

end(if)

- **Step 8:**

If $\rho_k > \rho_{end}$

Reduce ρ_k, reduce Δ to $\max\ [\frac{1}{2}\rho_k, \rho_{k+1}]$

set $k = k + 1$, go to step 3.

end(if).

- **Step 9:**

If $Ar = 1$

Calculate $f(\underline{x}_{opt} + \underline{p}_k)$

If $f(\underline{x}_{opt} + \underline{p}_k) < f(\underline{x}_{opt})$

overwrite $\underline{x}_{opt}$ by $\underline{x}_{opt} + \underline{p}_k$

end(if)

end(if)

output the final optimal point $\underline{x}_{opt}$, stop.

4.8 Convergence of LCOBYQA algorithm

LCOBYQA is a trust-region based algorithm. Therefore, the global convergence of this algorithm fits well within the convergence theory of trust-region algorithms. We do not present the theory in details, we just focus on the main ideas, because a rigorous derivation of the global convergence theory is found in (Conn et al., 2000). The theory of convergence is based on the following concepts

- The concept of validity
- The concept of sufficient decrease

First, we explain the concept of validity. We say that a model Q_k is valid in a ball $B(\underline{x}_{opt}, \delta)$, if the absolute error is bounded by a constant multiple of δ^2. This means

$$\mid f(\underline{x}) - Q_k(\underline{x}) \mid \leq c_1\delta^2, \quad \forall \underline{x} \in B(\underline{x}_{opt}, \delta), \tag{4.43}$$

for some constant c_1 inpendent of $\underline{x}$. This property implies that the model approximates the objective function accurately when the trust-region radius becomes small. An important characteristic of a valid model is that the gradient of the model coincides with the gradient of the objective function when the radius goes to zero. Indeed, if the model is valid, one can also show that

$$\parallel \nabla f(\underline{x}) - \nabla Q_k(\underline{x}) \parallel \leq c^2\delta, \ \forall \underline{x} \in B(\underline{x}_{opt}, \delta). \tag{4.44}$$

See (Conn et al., 2000).

The concepts of sufficient decrease corresponds to the minimum requirement about the decrease of the approximation to objective objective function in order to guarantee the convergence. The decrease must satisfy the following inequality

$$Q_k(\underline{x}_{opt}) - Q_k(\underline{x}_{opt} + \underline{p}_k) \geq \kappa 1 \parallel \nabla Q_k(\underline{x}_{opt}) \parallel \min(\frac{\parallel \nabla Q_k \parallel}{\beta_k}, \Delta_k), \ \forall \kappa 1, \tag{4.45}$$

where $\kappa 1$ is a constant in $(0, 1)$, and $\beta_k = 1 + \min_{\underline{x} \in B(\underline{x}_{opt}, \delta)} \parallel \nabla Q_k(\underline{x}) \parallel$. In addition to the above assumptionsin presented in equations (4.43), and (4.43), there are further assumptions to obtain the global convergence. These assumptions are:

A1: $f(\underline{x})$ is twice-continuosly differentiable in $\mathbb{R}^n$

A2: $f(\underline{x})$ is bounded below on $\mathbb{R}^n$

A3: the model Q_k is twice-continuosly differentiable in $\mathbb{R}^n$

A4: the Hessian of $f(\underline{x})$ is bounded on $\mathbb{R}^n$

A5: the Hessian of Q_k is bounded on $\mathbb{R}^n$

A6: $f(\underline{x})$ satisfies condition (4.43)

A7: the validity of Q_k in $B(\underline{x}_{opt}, \delta)$ may checked for each k and $\delta \geq 0$

A8: the model is valid after a finite number of improvement steps

Theorem 4.8.1. *If the assumptions A1 to A8 are satisfied, then every limit point $\underline{x}_\star$ of a sequence of iterations generated by LCOBYQA algorithm is a first order critical point.*

Proof: see (Conn et al., 2000).

4.8.1 Boundedness of the interpolation error

To apply the convergence theory developed above, we must show that the interpolation error is of the form given in expression (4.43). According to a general theorem formulated and demonstrated by Sauer and Xu (Sauer and Xu, 1995), our model is valid. Because the theorem by Sauer and Xu, gives a bound for the multivariate interpolation error when polynomials of degree $d \geq 1$ are used as interpolant functions.
We assume in this subsection that the objective function $f(\underline{x}), \underline{x} \in \mathbb{R}^n$ has its third derivatives that are bounded and continuous. Therefore, if $\underline{y}$ is any point in $\mathbb{R}^n$ and $\bar{\underline{d}}$ is any vector in $\mathbb{R}^n$ that has $\| \bar{\underline{d}} \| = 1$, then the function of one variable

$$\phi(\alpha) = f(\underline{x} + \alpha \bar{\underline{d}}), \ \alpha \in \mathbb{R}, \tag{4.46}$$

also has bounded and continuous third derivative. Further, there is a least nonnegative number ζ, independent of $\underline{y}$ and $\bar{\underline{d}}$, such that every function of this form has the property

$$| \phi'''(\alpha) | \leq \zeta, \ \alpha \in \mathbb{R}. \tag{4.47}$$

This value of ζ is suitable for the Interpolation error bound: Error$=| Q(\underline{x}) - f(\underline{x}) | < \frac{1}{6}\zeta \sum_{j=1}^{m} | \ell_j(\underline{x}) | \| \underline{x} - \underline{x}_j \|^3$, which stated and proved as a theorem.

Theorem 4.8.2. *Let the statement of the previous paragraph hold, and let the interpolation points be* $\underline{x}_i \in \mathbb{R}^n$, $i = 1, 2, \ldots, m$, *in any positions such that the system (3.11) define a unique quadratic polynomial* Q *from* $\mathbb{R}^n$ *to* $\mathbb{R}$. *Then, for every* $\underline{x} \in \mathbb{R}^n$, *the error of the quadratic model satisfies the condition*

$$| Q(\underline{x}) - f(\underline{x}) | < \frac{1}{6}\zeta \sum_{j=1}^{m} | \ell_j(\underline{x}) | \| \underline{x} - \underline{x}_j \|^3, \tag{4.48}$$

where the functions $\ell_j,\ j = 1, 2, \ldots, m$, *are the Lagrange functions of the interpolation points.*

Proof: We choose an arbitrary fixed point $\underline{y} \in \mathbb{R}^n$, and we derive a bound on $| Q(\underline{y}) - f(\underline{y}) |$. We used the Taylor series expansion of $f(\underline{x}),\ \underline{x} \in \mathbb{R}^n$, about the point $\underline{y}$. Specifically, we let $T(\underline{x}),\ \underline{x} \in \mathbb{R}^n$, be the quadratic polynomial that contains all the zero order, the first order and the second order terms of this expansion, and we consider the possibility of replacing the objective function f by f-T. The replacement would preserve all the third derivatives of the objective function, because T is quadratic polynomial. Therefore the given choice of ζ would remain valid. Furthermore, the quadratic model of f-T that is defined by the interpolation method would be Q-T. It follows that the error of the new quadratic model of the new objective function is f-Q as before. Therefore, when seeking about on $| Q(\underline{y}) - f(\underline{y}) |$ in terms of the third derivatives of the objective function, we assume without loss of generality that the function value $f(\underline{y})$, the gradient vector $\nabla f(\underline{y})$ and the second derivative matrix $\nabla^2 f(\underline{y})$ are all zero. Let j be an integer from $[1, m]$ such that $\underline{x}_j \neq \underline{y}$, let $\underline{\bar{d}}$ be the vector

$$\underline{\bar{d}} = (\underline{x}_j - \underline{y}) / \| \underline{y} - \underline{x}_j \|, \tag{4.49}$$

which has unit Euclidean length, and let $\phi(\alpha),\ \alpha \in \mathbb{R}$, be the function (4.46). The Taylor series with explicit remainder gives the formula

$$\phi(\alpha) = \phi(0) + \alpha\phi^{'}(0) + \frac{1}{2}\alpha^2\phi^{''}(0) + \frac{1}{6}\alpha^3\phi^{'''}(\xi), \quad \alpha \geq 0, \tag{4.50}$$

where ξ depends on α and in the interval $[0, \alpha]$. The value of $\phi(0), \phi^{'}(0)$ and $\phi^{''}(0)$ are all zero due to the assumptions of the previous paragraph, and we pick $\alpha = \| \underline{y} - \underline{x}_j \|$.

Thus, expressions (4.49), (4.46), (4.50) and (4.47) provide the bound

$$| f(\underline{x}_j) | = \frac{1}{6}\alpha^3 \mid \phi'''(\xi) | \leq \frac{1}{6}\zeta \parallel \underline{y} - \underline{x}_j \parallel^3, \tag{4.51}$$

which also holds without the middle part in the case $\underline{y} = \underline{x}_j$, because of the assumption $f(\underline{y}) = 0$. Using $f(\underline{y}) = 0$ again, we deduce from formula (3.28) and from inqueality (4.51) that the error $f(\underline{y}) - Q(\underline{y})$ has the property

$$| f(\underline{y}) - Q(\underline{y}) | = | Q(\underline{y}) | = | \sum_{j=1}^{m} f(\underline{x}_j)\ell_j(\underline{y}) | \leq \frac{1}{6}\zeta \sum_{j=1}^{m} \mid \ell_j(\underline{y}) \mid \parallel \underline{y} - \underline{x}_j \parallel^3 . \tag{4.52}$$

Therefore, because $\underline{y}$ is arbitrary, the theorem is true.□

Chapter 5

Numerical Results and Discussion

In this chapter, we compare the performance of LCOBYQA with that of other available model-based derivative-free algorithms. LCOBYQA software is tested on different test functions, unconstrained, simple bound constrained and linear constrained problems. Most of these test functions are non-convex. The comparison criteria are: the number of function evaluations that each algorithm takes to solve the problem, the final function values that each algorithm achieves, and the magnitude of the error in the approximate solution. We do not list the CPU time, because as mentioned in the definition of derivative-free optimization, the evaluation time is high. The numerical results discussed in this chapter are carried out on a pentium-4, 2.0 GHz, PC. On the other hand, most of available model-based derivative-free algorithms in the literature used either multi processors or multi work stations.

Firstly, LCOBYQA is tested on some unconstrained test problems. The test problems ARWHEAD, BDQRTIC (Conn et al., 1996), and CHROSEN (Toint, 1978) are used. The name ARWHEAD refers to a class of problems in which the nonzero elements of $\nabla^2 f(\underline{x})$ have an arrow head structure, $f(\underline{x})$ being the function

$$f(\underline{x}) = \sum_{i=1}^{n-1} \left\{ (x_i^2 + x_n^2)^2 - 4x_i + 3 \right\}, \ \underline{x} \in \mathbb{R}^n. \tag{5.1}$$

The BDQRTIC test problem has the objective function

$$f(\underline{x}) = \sum_{i=1}^{n-4} \left\{ (x_i^2 + 2x_{i+1}^2 + 3x_{i+2}^2 + 4x_{i+3}^2 + 5x_n^2)^2 - 4x_i + 3 \right\}, \ \underline{x} \in \mathbb{R}^n, \tag{5.2}$$

which is an extension of ARWHEAD that adds a band matrix of width seven to the previous arrow head structure of $\nabla^2 f(\underline{x})$. Moreover, the name CHROSEN denotes the chained Rosenbrock, $f(\underline{x})$ being the function

$$f(\underline{x}) = \sum_{i=2}^{n} \left[4(x_{i-1} - x_i^2)^2 + (1 - x_i)^2 \right], \ \underline{x} \in \mathbb{R}^n, \tag{5.3}$$

$\nabla^2 f(\underline{x})$ is a tridiagonal matrix. We see that all three objective functions are quadratric polynomials. In each case, we let the initial and final values of ρ to be $\rho_{beg} = 0.5$ and $\rho_{end} = 10^{-6}$. The initial vectors of variables are set to $\underline{e}$, $\underline{e}$ and $-\underline{e}$ for the three functions respectively, where $\underline{e}$ is the vector of ones in $\mathbb{R}^n$, and where $n = 10,\ n = 15,\ n = 20$ and $n = 25$, were tried. The number of evaluations of $f(\underline{x})$ for LCOBYQA, UOBYQA, and COBYLA, are reported in Table 5.1. We are not used the other criterias because they are not presented in the literature. Table 5.1 shows that our algorithm is atractive in comarison with its competitors, namely COBYLA and UOBYQA, except that the result of BDQRTIC, where we observe that the performance of UOBYQA is superior to that of LCOBYQA. Further work is needed to justify this observation.

The LOBYQA is also compared to CONDOR and DFO. The following unconstrained test problems are used. The POWER function (Bongartz et al., 1995), which has the objective function

$$f(\underline{x}) = \sum_{i=1}^{n} (ix_i)^2, \ \underline{x} \in \mathbb{R}^n, \tag{5.4}$$

with the initial point $\underline{x}_0 = \underline{e}$. The DQDRTIC test problem that has the objective function

$$f(\underline{x}) = \sum_{i=1}^{n-2} (x_i^2 + cx_{i+1}^2 + dx_{i+2}^2)^2, \quad c = 100, \quad d = 100, \ \underline{x} \in \mathbb{R}^n, \tag{5.5}$$

with starting point $\underline{x}_0 = 3\underline{e}$, and VARDIM (Buckley and Jenning, 1989), which has the objective function

$$f(\underline{x}) = \sum_{l=1}^{n} (x_l - 1)^2 + \left\{ \sum_{l=1}^{n} l(x_l - 1) \right\}^2 + \left\{ \sum_{l=1}^{n} l(x_l - 1) \right\}^4, \ \underline{x} \in \mathbb{R}^n, \tag{5.6}$$

Table 5.1: *Number of function evaluations for LCOBYQA, COBYLA and UOBYQA.*

Test problem	LCOBYQA	COBYLA	UOBYQA
ARWHEAD, n=10	184	280	219
ARWHEAD, n=15	224	522	458
ARWHEAD, n=20	368	678	837
ARWHEAD, n=25	457	900	1320
BDQRTIC, n=10	660	1106	434
BDQRTIC, n=15	1273	2323	843
BDQRTIC, n=20	1764	3616	1541
BDQRTIC, n=25	2369	5619	2302
CHROSEN, n=10	411	4661	454
CHROSEN, n=15	701	6935	1064
CHROSEN, n=20	977	8912	1897
CHROSEN, n=25	1216	10861	2565

which takes its least value of zero at $\underline{x} = \underline{e}$, the vector of ones. Analytic differentation of VARDIM gives the second derivative matrix

$$\nabla^2 f(\underline{x}) = 2I + \left[2 + 12\left\{\sum_{l=1}^{n} l(x_l - 1)\right\}^2\right] \Phi, \tag{5.7}$$

where I is the $n \times n$ unit matrix and Φ is rank one matrix that has the elements $\Phi_{ij} = ij,\ 1 \leq i, j \leq n$.

Table 5.2 shows the number of function evaluations and the final function values for POWER, DQDRTIC and VARDIM test functions. As before, we are not used the third criteria because, it is not presented in the literature. Table 5.2 shows that the performance of LCOBYQA is far superior to that of the other two algorithms, namely CONDOR and DFO. However, LCOBYQA does not do as well as DFO on VANDIM. This observation

needs further work to justify.

Table 5.2: *Comparative results between LCOBYQA, CONDOR and DFO.*

	number of function evaluations			**final function values**		
NAME, DIM	LCOBYQA	CONDOR	DFO	LCOBYQA	CONDOR	DFO
DQDRTIN, n=10	28	201	403	2.1777e-25	2.0929e-18	1.6260e-20
POWER, n=10	28	550	206	6.6831e-26	9.5433e-7	2.0582e-7
VARDIM, n=10	2517	2686	2061	3.15423e-10	2.177e-25	1.626e-20

LCOBYQA is also tested and compared to NEWUOA on different unconstrained test functions. We used the ARWHEAD test problem (5.3), CHROSEN probem (5.5), the PENALTY1, PENALTY2, PENALTY3 problems (Buckley and Jenning, 1989). The PENALTY1 objective function is:

$$f(\underline{x}) = 10^{-5}\sum_{i=1}^{n}(x_i-1)^2 + \left\{\frac{1}{4} - \sum_{i=1}^{n} x_i^2\right\}^2, \ \underline{x} \in \mathbb{R}^n, \tag{5.8}$$

with the starting point $(\underline{x})_i = i, \ i = 1, 2, \ldots, n$. The PENALTY2 test problem objective function is:

$$f(\underline{x}) = \sum_{i=2}^{n}\left\{(e^{x_{i-1}/10} + e^{x_i/10} - e^{(i-1)/10} - e^{i/10})^2 + (e^{x_i/10} - e^{-1/10})^2\right\} +$$

$$\left\{1 - \sum_{i=1}^{n}(n-i+1)x_i^2\right\}^2 + (x_1 - \frac{1}{5}), \ \underline{x} \in \mathbb{R}^n, \tag{5.9}$$

with the starting point $\underline{x} = \frac{1}{2}\underline{e} \in \mathbb{R}^n$. The PENALTY3 test problem objective function is:

$$f(\underline{x}) = 10^{-3}(1 + Re^{x_n} + Se^{x_{n-1}} + RS) + \left\{\sum_{i=1}^{n}(x_i^2 - n)\right\}^2 + \sum_{i=1}^{n/2}(x_i - 1)^2, \ \underline{x} \in \mathbb{R}^n, \tag{5.10}$$

where n is an even number and where R and S are the sums

$$R = \sum_{i=1}^{n-2}(x_i + 2x_{i+1} + 10x_{i+2} - 1)^2 \tag{5.11}$$

$$\text{and} \quad S = \sum_{i=1}^{n-2}(2x_i + x_{i+1} - 3)^2. \tag{5.12}$$

The starting point is the zero vector. We set $\rho_{end} = 10^{-6}$ in each case, while ρ_{beg} is given the values $0.5,\ 0.5,\ 1.0,\ 0.1$ and 0.1 for ARWHEAD, CHROSEN, PELANTY1, PELANTY2 and PELANTY3, respectively. Table 5.3 shows the number of function evaluations of LCOBYQA and NEWUOA for the 5 test functions. The star in Table 5.3 indicates that the CPU time is very long, so the problem is not tried. In this Table, we observe that the performance the two algorithms is similar, for example the results of LCOBYQA on ARWHEAD and PENALTY2 test problems are better than NEWUOA, but the results of NEWUOA on CHROSEN and PENALTY3 are better than LCOBYQA. Generally we can say that the results of the two algorithm are similar, which is not surprising since they are based on the same idea, namely the least Frobenius norm method.

The LCOBYQA software is also tested on the following trignomitric sum of squares objective function (TRIGSSQS)

$$f(\underline{x}) = \sum_{i=1}^{2n} \left[b_i - \sum_{j}^{n} (S_{ij} sin(\theta_j x_j) - C_{ij} cos(\theta_j x_j)) \right]^2 ,\ \underline{x} \in \mathbb{R}^n. \tag{5.13}$$

The way of generating the parameters of this function is taken from Fletcher and Powell (Fletcher and Powell, 1963). The elements of $2n \times n$ matrices S and C are random integers from the interval $[-100, 100]$, and each factor θ_j is sampled from the logarithmic distribution on [0.1,1]. The parameters $b_i,\ i = 1, 2, \ldots, 2n$ are defined by the equation $f(\underline{x}^\star) = 0$, where $\underline{x}^\star$ has the components $x_j^\star = \hat{x}_j^\star/\theta_j,\ j = 1, 2, \ldots, n$, where each $\hat{x}_j^\star$ is picked from the uniform distribution on $[-\pi, \pi]$. The starting point $\underline{x}_0$ has the components $(\hat{x}_j^\star + 0.1\hat{y}_j^\star)/\theta_j,\ j = 1, 2, \ldots, n$, where every $\hat{y}_j^\star$ is also taken from the randam distribution $[-\pi, \pi]$. There are three remarks to to be made about this objective function

1- Because the number of terms in the sum of squares is equal to the number of variables, it often happens that the Hessian martix is ill-conditioned around $\underline{x}^\star$.

2- Because $f(\underline{x})$ is periodic, it has many saddle points and maxima.

3- The advantage of using random numbers is that it is easy to generate many different objective functions.

Using this function it is possible to cover different type of problems. The results of LCOBYQA software for some of these cases with four values of n, and the parameters $\rho_{beg} = 10^{-1}$ and

Table 5.3: *Number of function evaluations on LCOBYQA and NEWUOA.*

Test problem	LCOBYQA	NEWUOA
ARWHEAD, n=20	321	404
ARWHEAD, n=40	1107	1497
ARWHEAD, n=80	2491	3287
ARWHEAD, n=160	8453	8504
CHROSEN, n=20	818	845
CHROSEN, n=40	2042	1876
CHROSEN, n=80	4852	4314
CHROSEN, n=160	$\star$	9875
PENALTY1, n=20	7507	7476
PENALTY1, n=40	16704	14370
PENALTY1, n=80	27407	32390
PENALTY1, n=160	$\star$	72519
PENALTY2, n=20	1612	2443
PENALTY2, n=40	5354	5703
PENALTY2, n=80	13475	$\star$
PENALTY2, n=160	$\star$	$\star$
PENALTY3, n=20	4337	3219
PENALTY3, n=40	14221	16589
PENALTY3, n=80	36039	136902
PENALTY3, n=160	$\star$	$\star$

$\rho_{end} = 10^{-6}$, are reported in Table 5.4. Each row of Table 5.4 gives n, the averages number of function evaluations (nof) for five different test problems and the greatest value of $\| \underline{x}_f - \underline{x}^\star \|_\infty$, where $\underline{x}_f$ is the final vector of variables calculated by the software, and $\underline{x}^\star$ is the optimal solution. Table 5.4 shows that the accuracy of the results is not so excellent, specially for problems with a large number of variables, but it reasonable. This is because the objective function is too difficult. Further work is needed to improve the accurancy of these results.

Table 5.4: *Averages for LCOBYQA applied to 5 versions of TRIGSSQS*

n	nof	$\| \underline{x}_f - \underline{x}^\star \|_\infty$
10	378.3	2.2897e-5
20	2100.6	0.0503
30	4646.2	0.1437
40	14648.6	0.5068

LCOBYQA algorithm is applied on some functions with discontinuous first derivatives. We use the TRIGSABS function which has the form

$$f(\underline{x}) = \sum_{i=1}^{2n} \mid b_i - \sum_{j=1}^{n} (S_{ij} sin(\theta_j x_j) - C_{ij} cos(\theta_j x_j)) \mid, \ \underline{x} \in \mathbb{R}^n. \tag{5.14}$$

The parameters $b_i,\ S_{ij},\ C_{ij}$ and the initial vector $\underline{x}_0$, are generated randomly as done for problem (5.13), except that we employ the scaling factors $\theta_j = 1,\ j = 1, 2, \ldots, n$. Different radom numbers provide five test problems for each n as before. We retain the parameters $\rho_{beg},\ \rho_{end}$ as in equation (5.16). Each row of Table 5.5 gives the dimession of the problem, n, the averages number of function evaluations (nof) for five different test problems and the greatest value of $\| \underline{x}_f - \underline{x}^\star \|_\infty$, where $\underline{x}_f$ is the final vector of variables calculated by the software, and $\underline{x}^\star$ is the optimal solution.Table 5.5 shows that the accuracy of the results is not so excellent, specially for problems with a large number of variables, but it reasonable. This is because the algorthim is originally designed for smooth functions.

LCOBYQA software is also tested on more than 50 problems with simple bounds and linear constraints. These problems are taken from Hock-Schittkowski-Collection (Hock and Schittkowski,

Table 5.5: *Averages for LCOBYQA applied to 5 versions of TRIGSABS*

n	nof	$\| \underline{x}_f - \underline{x}^\star \|_\infty$
5	162	6.3530e-5
10	701.5	0.1001
15	2002.5	0.12
20	2115	0.2

1981) and other sources. We set $\rho_{end} = 10^{-6}$ in each case, while ρ_{beg} is given the values 0.1 for each problem. Table 5.6 shows the results of some of these problems. Each row of this Table gives the name, dimession, n, the type of the constrants, and the number of function evaluations (nof), for each problem. The results obtained by our software for these problems are accurate and do agree with the results reported in Hock-Schittkowski-Collection (Hock and Schittkowski, 1981).

LCOBYQA software is tested and compared with CONDOR, DFO and COBYLA using the following test problems.
The first test problem is HS038 (Hock and Schittkowski, 1981), which has simple bound constraints. The objective function is

$$f(\underline{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1\left[(x_2 - 1)^2 + (x_4 - 1)^2\right] + 19.8(x_2 - 1)(x_4 - 1), \tag{5.15}$$

subject to

$$-10 \leq x_i \leq 10, \quad i = 1, 2, \ldots, 4,$$

with starting point $(-3, -1, -3, -1)^T$. The optimal point is $(1, 1, 1, 1)^T$. The results are given in table 5.7 below. Table 5.7 shows the outperformance of LCOBYQA compared to CONDOR, DFO and COBYLA.

Finally, LCOBYQA is also compared to CONDOR, DFO and COBYLA on the following linear constrained problems:

Table 5.6: *Results of test functions in Hock-Schittkowski-Collection*

NAME, DIM	type of constraints	nof	final function value
HS 01, 2	linear	273	9.90030e-13
HS 02, 2	linear	534	4.941
HS 03, 2	linear	9	1.3332e-33
HS 04, 2	linear	12	2.667
HS 05, 2	bound	30	-1.913
HS 09, 2,	linear	20	-.5000
HS 21, 2	linear	13	-99.9600
HS 24, 2	linear	13	-1.0000
HS 28, 3	linear	52	5.51140e-18
HS 35, 3	linear	84	0.1117
HS 36, 3	linear	20	-3.3000e+3
HS 37, 3	linear	113	-3.45600e+3
HS 38, 4	bound	174	2.8731e-10
HS 44, 4	linear	15	-15
HS 45, 5	bound	16	1.0000
HS 48, 5	linear	166	1.34710e-14
HS 50, 5	linear	107	1.0913e-12
HS 51, 5	linear	87	2.0866e-11
HS 76, 4	linear	16	-4.6818
HS 110, 10	bound	351	-45.7785

Table 5.7: *Comparative results between LCOBYQA, CONDOR, DFO and COBYLA*

	number of function evaluations				final function value			
NAME, DIM	LCOBYQA	CONDOR	DFO	COBYLA	LCOBYQA	CONDOR	DFO	COBYLA
HS038 n=4	174	311	535	382	2.8731e-10	7.8251e-13	1.6583e-7	7.8770e+0

The $HS044$ test problem:

$$\min_{\underline{x}\in\mathbb{R}^4} f(\underline{x}) = x_1 - x_2 - x_3 - x_1x_3 + x_1x_4 + x_2x_3 - x_2x_4 \tag{5.16}$$

subject to:

$$-x_1 - 2x_2 \geq -8$$

$$-4x_1 - x_2 \geq -12$$

$$-3x_1 - 4x_2 \geq -12$$

with the starting point $(0, 0, 0, 0)^T$.

The $HS076$ test problem:

$$\min_{\underline{x}\in\mathbb{R}^4} f(\underline{x}) = x_1^2 + 0.5x_2^2 + x_3^2 + 0.5x_4^2 - x_1x_3 + x_3x_4 - x_1 - 3x_2 + x_3 - x_4 \tag{5.18}$$

Subject to:

$$\begin{aligned} -x_1 - 2x_2 - x_3 - x_4 &\geq -5 \\ -3x_1 - x_2 - 2x_3 + x_4 &\geq -4 \\ x_2 + 4x_3 &\geq -4 \\ x_i \geq 0,\ i &= 1, 2, 3, 4 \end{aligned} \tag{5.19}$$

with starting point $(0.5, 0.5, 0.5, 0.5)^T$.

The results of these test functions is given in Table 5.8 below.

Table 5.8: *Comparative Results Between LCOBYQA, CONDOR, DFO and COBYLA*

	number of function evaluations				final function value			
NAME, DIM	LCOBYQA	CONDOR	DFO	COBYLA	LCOBYQA	CONDOR	DFO	COBYLA
HS044, n=4	15	23	26	45	-15	-15	-15	-15
HS076, n=4	16	21	29	76	-4.6818	-4.6818	-4.6818	-4.6818

From this Table, we see that LCOBYQA perform very well compared to CONDOR, DFO and COBYLA.

Chapter 6

Conclusion and Future Outlook

Based on Powell's algorithms (NEWUOA, BOBYQA), we have developed a successesful new derivative-free algorithm named LCOBYQA, to solve linearly constrained optimization problems. The algorithm is based on quadratic interpolation. It constructs a quadratic model of the objective function from a few data, and uses the remaining freedom in the model to minimize the Frobenius norm of the Hessiam matrix of the change to the model.

In chapter 5, we tested our algorithm (LCOBYQA) on various test problems, most of which are nonconvex problems. Firstly, LCOBYQA is tested on unconstrained test functions. Table 5.1 shows that our algorithm is atractive in comarison with its competitors, namely COBYLA and UOBYQA. However, LCOBYQA does not do as well as UOBYQA on BDQRTIC. LCOBYQAal is also compared to CONDOR and DFO algorithms. Table 5.2 shows that the number of function evaluations in LCOBYQA is less than the number of function evaluations in its competitors, except that of VARDIM, where the result of DFO is better . LCOBYQA is also compared to NEWUOA algorithm, Table 5.3 shows that the results of algorithms are approximately similar, which is not surprising since they are based on the same idea, namely the least Frobenius norm method. Secondly, LCOBYQA is applied to the trignomitric sum of squares test functions and also to some functions with discontinuous first derivatives. Tables 5.4 and 5.5 show that the accuracy of the results is not so excellent, specially for problems with a large number of variables, but it reasonable. Lastly, LCOBYQA is tested on simple bound and linearly constrained problems. Table 5.6 shows the name, dimension, number of function evaluations and the final function

values for some test functions taken from HS collection. In Tables 5.7 and 5.8, LCOBYQA is compared to CONDOR, DFO and COBYQA. The Tables show that LCOBYQA is much more powerfull tool. Hence, the results obtained by the algorithm prove its efficiency and shows that it competes favourably against other model based algorithms.

It is important to report that all the numerical results discussed in chapter 5 are carred out on a pentium-4, 2.0 GHz, PC. On the other hand, most of available model-based derivative-free algorithms in the literature used either multi processors or mlti work stations.

The work reported in this thesis is considered as a starting piont for the solution of the large class of constrained optimization problems. For future work, in this regard, we suggwst the folowing:

Firstly, the inclusion of quadratic, generally nonlinear and difficult constraints in our work is of interest. In order to handle quadratic and general constraints, we can use the augmented Lagrangian method or the exactly penalty function method or the filter method. For the difficult constraints, the idea is to use the interpolation to construct the constraints in the same way as the interpolation of the objective function, as applied in COBYLA algorithm (Powell, 1994).

The number of interpolation points needed to construct the first quadratic model in our algorithm is $m = 2n+1$. It is possible to construct a quadratic model using less points according to the inequality $n + 2 \leq m \leq 2n + 1$, as used in NEWUOA (Powell, 2006) and BOBYQA (Powell, 2009) algorithms.

Currently, the trust-region is a simple ball. It would be interesting to have a trust-region which reflects the underlying geometry of the model and not give too much weight to certain direction, for example, we can use other norms like semi-norm, as applied in (Powell, 2010).

The code of the algorithm is a complete MATLAB package, there is no call to any external, unavailable libraries, which makes the algorithm relatively fast. To increase the speed of the algorithm further, it would be interesting to use the technique of parallelsim in coding, as used in CONDOR (Frank and Bersini, 2005).

It has been observed that the algorithm converges faster when RESCUE procedure of section 6 chapter 3 happens to be called so early during first few iterations. Using this observation, it is possible to solve the ARWHEAD test function up to 900 variables using a single processor.

This is considered amazing. Further work is needed to justify this.

Finally, it is clear that considerable further numerical experience is needed to develop a robust and efficeint algorithm. Specialy, it would be important to use the algorithm to solve real application where the objective function is very expensive.

References

Bongartz, I., Conn, Gould, and Toint (1995). CUTE: constrained and unconstrained testing environment. *ACM Transaction on Mathematical Software*, 21(1):123–160.

Broyden, C. G., Denni, S. J. E., and Mor$\acute{e}$, J. (1973). On the local and supperlinear convergence of quasi-newton method. *J. Inst. Appli*, 16:245.

Buckley, A. G. and Jenning, L. S. (1989). Test functions for unconstrained minimization. Technical report, CS-3, Dalhousie University, Canada.

Byrd, Richard, H., Gould, N. I. M., Nocedal, Jorge, and Waltz (2002). An active-set algorithm for nonlinear programming using linear programming and equality constrained subproblem. Technical report, RAL-TR-2002-2003.

Colson, Beno$\hat{i}$t, and Toint, P. L. (2003). Optimizing partially separable functions without derivatives. Technical report, University of Namur, Department of Mathematics.

Colson, Beno$\hat{i}$t, C. G., Dennis, J. E., and Mor$\acute{e}$, J. (2004). Trust region algorithm for derivative free optimization and nonlinear bilevel programming. *Springer Verlag*, 4(2):85–88.

Conn, Gould, and Toint (2000). *Trust Region Methods*. SIAM.

Conn, Sheinberg, and Toint (1996). An algorithm using quadratic interpolation for unconstrained derivative-free optimization. *Nonlinear Optimization and Application, G. Di. Pillo and F. Giannessi (eds), New Youk, Plenium*, pages 27–47.

Conn, Sheinberg, and Toint (1997a). On the convergence of the derivative free methods for unconstrained optimization. *In A. Iserles and M. Buhman (eds), Approximation Theory and Optimization: Tributes to M. J. P. Powell, Cambradge University Press, Cambradge, Uk*, pages 83–108.

Conn, Sheinberg, and Toint (1997b). Recent progress in unconstrained nonlinear optimization without derivative. *Mathematical Programming*, 79(B):397–414.

Conn, Sheinberg, and Toint (1998). A derivative free optimization (DFO) algorithm. Technical report, Report 98/11.

Conn, Sheinberg, and Vicente (2009). *Introduction to Derivative Free Optimization*. SIAM Publications (Philadephia).

Dennis and Schabel (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, England, Cliffs, NJ, Reprinted by SIAM Bublications.

Dennis and Schabel (2000). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. SIAM.

Dong and Shaoman (2002). Methods for constrained opimization. *Springer*, 18:project.

Faul, A. C. (2001). *Iterative Techniques for Radial Basis Function Interpolation*. Phd, University of Cambridge.

Fletcher, R. (1987). *Practical Methods of Optimization*. 2nd Ed., John Wiley & Sons.

Fletcher, R. and Powell, M. J. D. (1963). A rapidly convergent descent method for minimization. *Comput. J.*, 8:33–41.

Forsgen and Murray (1997). Newton method for large scale linear inequality consrained minimization. *SIAM*, 181:162–176.

Frank and Bersini, H. (2005). CONDOR, a new prallel constrained extension of powell's uobyqa algorithm: Expremental results and comparision with dfo algorithm. *Journal of Computational and Applied Mathematics*, 181:157–175.

Gay and David, M. (1984). Lecture note in mathematics. *Springer Verlag*, 1066:74–105.

Gill, Philip, Murray, Sounder, and Wright (1991). Inertia-controlling methods for general quadratic programming. *SIAM*, 33:1–36.

Gill, P. E., Murray, and Wright (1988). *Practical Optimization*. Academic Press.

Gould, N. I. M., Hribar, M. E., and Nocedal, J. (1998). On the solution of equatilty constrained quadratic programming problems arising in optimization. Technical report, Oxfordshire OX1100Qx.

Gould, N. I. M., Orban, D., and Toint, P. L. (2004). Numerical methods for large-scale nonlinear optimization. Technical report, Oxfordshire OX1100Qx, Uk, CCLRC Rutherfor Appleton Laboratory.

Gould, N. I. M. and Toint, P. L. (2001). An iterative working-set method for large scale nonconvex quadratic programming. *RAL-TR*, pages 200–260.

Hashim, M. H. (1995). *An Algorithm for General Quadratic Programming*. Phd, Faculty of Mathematical Sciences, University of Khartoum, Sudan.

Hock, W. and Schittkowski, k. (1981). Test example for nonlinear programming codes. Lecture Notes en Economics and Mathematical Sysrems, 187.

Igor, Nash, and Sofer (2009). *Linear and Nonlinear Programming*. SIAM.

Ju, Changyu, Wang, and Jiguoyo (2004). Combining trust region and linesearch algorithm for equality constrained optimization. *Applied Mathematics and Computational*, 14(1-2):123–136.

Ju, Liang, and Zhang (2001). A robust trust region method for nonlinear optimization with inequality constraints. *Applied Mathematics and Computation*, 176:688–699.

Levenberg, K. (1944). A method for the solution of certain nonlinear problem is least squares. *Quartely of Applied Mathematics*, 2:164–168.

Lewis, R. M., Torczon, V., and Trosset, M. W. (2000). *Direct search Methods: Then and now*. Journal of Computational and Applied Mathematics.

Maquardt, D. W. (1963). An algorithm for least square estimation of nonlinear parameter. *SIAM Journal*, 11:431–441.

More and Sorensen (1983). a trust region step. *SIAM Journal on Scientific and Computing*, 4:553–572.

Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. 2nd, Eds, Springer Verlag.

Oeuvray, R. (2005). *Trust Region Methods Based on Radial Functions with Application to Biomedical Imaging*. Phd, EPFL, Lausanne.

Powell, M. J. D. (1964). An efficient method for finding the minimum of a function of several variables without calculating the derivatives. *Computer Journal*, 7:155–162.

Powell, M. J. D. (1994). A direct search optimization method that model the objective function and constrained functions by linear interpolation. *In advances in Optimization and Numericval Analysis, Proceeding of the Sixth Workshop on Optimization and Numericval Analysis, Oaxaca, Mexico, S. Gomez and J. P. Hennart, eds.*, 275:15–67.

Powell, M. J. D. (1998). Direct search algorithms for optimization calculations. *Acta Numerica*, 7:287–336.

Powell, M. J. D. (2001). On the Lagrange functions of quadratic models that are define by interpolation, opti. meth. softw. *J. Inst. Appli*, 16:289–309.

Powell, M. J. D. (2002). UOBYQA: Unconsrained optimization by quadratic approximation. *Mathematical Programming*, 92:555–582.

Powell, M. J. D. (2003). On the trust region methods for unconstrained minimization without derivative. *Math. Program.*, 7:605–623.

Powell, M. J. D. (2004a). Least Frobenius norm updating for quadratic models that satisfy interpolation conditions. *Math. Program.*, 100:183–215.

Powell, M. J. D. (2004b). On updating the inverse of KKT matrix, in numerical linear algebra and optimization. *In Numerical Linear Algebra and Optimization*, 1:56–78.

Powell, M. J. D. (2006). The NEWUOA software for unconstrained optimization without derivative. *In Large Scale Nonlinear Optimization, eds. G. Di pillo and M. Roma, Springer(NEW YORK)*, pages 255–297.

Powell, M. J. D. (2008). Development of NEWUOA for minimization without derivatives. *IMA J. Numer. Anal.*, 28:649–664.

Powell, M. J. D. (2009). The BOBYQA algorithm for bound constrained optimization without derivative. Technical report, 2009/NA06, CMS University of Cambridge.

Powell, M. J. D. (2010). Beyond symmetric broyden for updating quadratic models in minimization without derivatives. Technical report, No DEMTP 2010/NA04, University of Cambridge.

Richard, Byrd, H., Schnabel, B., Gerald, and Shultz, A. (1987). A trust region algorithm for nonlinear constrained optimization. *SIAM Journal on Numerical Alalysis*, 24(5):1152–1170.

Sauer, T. and Xu, Y. (1995). On multivariate Lagrange interpolation. *Mathematics of Computation*, 64:1147–1170.

Sheinberg and Vicente, L. (2006). Error estimate and poisedness in multivariate polynomial interpolation. Technical report, IBM T. J. Watson research.

Sheinberg and Vicente, L. (2007). Geometry of interpolation set in derivative-Free optimization. *Mathematical Programming*, A(111):141–172.

Sheinberg and Vicente, L. (2008). Geometry of sample sets in derivative free optimization: Polynomail regression and underdetermined interpolation. *IMA. J Numer. Anal.*, 28:721–748.

Sheinberg, K. (2000). Derivative free optimization method. Technical report, Preprint, IBM, T. J. WATSON Research Centre.

Toint, P. L. (1978). Some numerical results using a sparse matrix updating formula in unconstrained optimization. *Math. Comp.*, 32:839–859.

Vadi and Avi (1985). A trust region algorithm for equality constrained minimization: Convergence properties and implementation. *Math. Comp.*, 22(3):575–591.

Winfield, D. (1969). *Function and Functional Optimization by Interpolation in Data Table*. Phd, Harvard University, Cambridge, USA.

Winfield, D. (1973). Function minimization by interpolation in data table. *Journal of Institute of Mathematics and its Application*, 12:339–347.

XU, D. C., Han, J. Y., and Chen, Z. W. (2004). Nonmonotone trust region method for nonlinear programming with general constraained and simple bounds. *Journal of Institute of Mathematics and its Application*, 122(1):185–206.

Ying-Ji, Ke, C. Z., Shao-Jian, and Ying-Ahou (2007). A trust region method by acive set strategy for general nonlinear optimization. *Journal of Institute of Mathematics and its Application*, 54:229–241.

Zhibin, Z. and Binliang, Z. (2006). A general gradient method for linear constrained optimization with supper linear convergence. *Journal of Applied Science*, 6(5):1085–1089.

Appendix A

Proofs of some Assertions

A.1 Proof of theorem 2.2.1

Since A^TA is Hermition, there exists an eigendecomposition $A^TA = Q\Lambda Q^T$ with Q a Unitary matrix (the columns are eigenvectors), and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ a diagonal matrix containing the eigenvalues, which must be real. Note that all $\lambda_i \geq 0, i = 1, 2, ..n$, since if one, say λ, were negative, we could take $\underline{q}$ as its eigenvector and we get the contradiction $0 \leq \| A\underline{q} \|_2^2 = \underline{q}^T A^T A\underline{q} = \underline{q}^T \lambda \underline{q} = \lambda \| \underline{q} \|_2^2 < 0$.

Therefore,

$$\| A \|_2 = \max_{\underline{x} \neq 0} \frac{\| A\underline{x} \|_2}{\| \underline{x} \|_2} = \max_{\underline{x} \neq 0} \frac{(\underline{x}^T A^T A \underline{x})^{\frac{1}{2}}}{\| \underline{x} \|_2} = \max_{\underline{x} \neq 0} \frac{(\underline{x}^T Q \Lambda Q^T \underline{x})^{\frac{1}{2}}}{\| \underline{x} \|_2} =$$

$$\max_{x \neq 0} \frac{((Q^T\underline{x})^T \Lambda Q^T \underline{x})^{\frac{1}{2}}}{\| Q^T \underline{x} \|_2} = \max_{\underline{y} \neq 0} \frac{(\underline{y}^T \Lambda \underline{y})^{\frac{1}{2}}}{\| \underline{y} \|_2} = \max_{\underline{y} \neq 0} \sqrt{\frac{\sum \lambda_i y_i^2}{\sum y_i^2}} \leq \max_{\underline{y} \neq 0} \sqrt{\omega} \sqrt{\frac{\sum y_i^2}{\sum y_i^2}} = \sqrt{\omega} \quad \text{(A.1)}$$

which is attainable by choosing $\underline{y}$ to be appropriate column of identity matrix.

A.2 Proof of theorem 3.4.1

H^+ is define to be the inverse of the symmetric matrix whose t-th column is $\underline{w}^+$ and whose other columns are the vectors

$$\underline{v}_j = H^{-1}\underline{e}_j + (\underline{e}_j^T \underline{w}_t^+ - \underline{e}_j^T H^{-1} \underline{e}_t)\underline{e}_j, \; j \in \{1, 2, \ldots, n+m+1\} \setminus \{t\}\,. \quad \text{(A.2)}$$

Therefore, letting H^+ be the matrix (3.41), is it sufficient to establish $H^+\underline{w}^+ = \underline{e}$ and $H^+\underline{v}_t = \underline{e}$, $j \neq t$. Because equation (3.42) shows that β_t^+ and τ_t^+ are the scalar products $(\underline{e}-H\underline{w}_t^+)^T\underline{w}_t^+$ and $\underline{e}^T H\underline{w}_t^+$, respectively, formula (3.41) achieves the conditions

$$\begin{aligned} H^+\underline{w}_t^+ &= H\underline{w}_t^+ + \frac{1}{\sigma^+}\left[\alpha_t^+\beta_t^+(\underline{e}_t - H\underline{w}_t^+) - \beta_t^+\tau_t^+ H\underline{e}_t + \tau_t^+\left\{\beta_t^+ H\underline{e}_t + \tau_t^+(\underline{e}_t - H\underline{w}_t^+)\right\}\right] \\ &= H\underline{w}_t^+ + \frac{1}{\sigma^+}\left\{\alpha_t^+\beta_t^+ + \tau_t^2\right\}(\underline{e}_t - H\underline{w}_t^+) = \underline{e}_t, \end{aligned} \tag{A.3}$$

the last equation due to the definition (3.42) of σ_t^+. It follows that, if j is any integer from $[1, n+m+1]$ that is different from t, then it remains to prove $H^+\underline{v}_j = \underline{e}_j$.

Formula (3.41), $j \neq t$ and the symmetry of H^{-1} provide the identity

$$H^+(H^{-1}\underline{e}_j) = \underline{e}_j + \frac{(\underline{e}_t - H\underline{w}_t^+)^T H^{-1}\underline{e}_j}{\sigma_t^+}\left[\alpha_t^+(\underline{e}_t - H\underline{w}_t^+) + \tau_t^+ H\underline{e}_t\right]. \tag{A.4}$$

Moreover, because the scalar products $(\underline{e}_t - H\underline{w}_t^+)^T\underline{e}_t$ and $\underline{e}_t^T H\underline{e}_t$ take the values $1-\tau_t^+$ and α_t^+, formula (3.41) also gives the property

$$\begin{aligned} H^+\underline{e}_t = H\underline{e}_t + \frac{1}{\sigma^+}&\left[\alpha_t^+(1-\tau_t^+)(\underline{e}_t - H\underline{w}_t^+) - \beta_t^+\alpha_t^+ H\underline{e}_t + \tau_t^+\left\{(1-\tau_t^+)H\underline{e}_t + \alpha_t^+(\underline{e}_t - H\underline{w}_t^+)\right\}\right] \\ &= \frac{1}{\sigma^+}\left[\alpha_t^+(\underline{e}_t - H\underline{w}_t^+) + \tau_t^+ H\underline{e}_t\right]. \end{aligned} \tag{A.5}$$

The numerator in the expression (A.3) has the value $-(\underline{e}_t^T\underline{w}_t^+ - \underline{e}_t^T H^{-1}\underline{e}_t)$. Therefore equations (A.1), (A.3) and (A.4) imply the identity $H^+\underline{v}_j = \underline{e}_j$, which complete the proof $\square$.

Appendix B

Codes

This procedure provides strictly feasible point

B.1 Procedure Phase one

```
function x0=phase1(A,b)
[m,n]=size(A);
p=n;
delta=.1;
delta=2*delta;
pp=m;
s=0;
B1=zeros(m,1);
B2=B1;
M1=zeros(m,1);
M2=M1;
M3=M1;
M4=M1;
s1=0;
s2=0;
```

```
s3=0;
for i=1:m
    if b(i)<0
        b(i)=-b(i);
        A(i,:)=-A(i,:);
        B1(i,1)=1;
        M1=[M1 B1];
        B1(i,1)=0;
        s=s+1;
     else
        B1(i,1)=1;
        B2(i,1)=-1;
        M2=[M2 B2];
        M4=[M4 B1];
        B1(i,1)=0;
        B2(i,1)=0;
        s3=s3+1;
    end
end
if s>0
    M11=M1(:,2:s+1);
end
if s3>0
    M22=M2(:,2:s3+1);
    M44=M4(:,2:s3+1);
end
if s>0
      AA=[A M11];
else
    AA=A;
end
if s3>0
```

```
   AA=[AA M22];
   AA=[AA M44];
end
ss=s3;
[m1,n1]=size(AA);
xx=ones(1,ss);
xxx=zeros(1,n1-ss);
y=[xxx xx];
A0=[y;AA];
b0=[0;b];
AAA=[A0 b0];
[m2,n2]=size(AAA);
for i=n+s3+s+1:n2-1
  t=2;
  for j=2:m2
      if AAA(j,i)==1
          t=j;
      end
  end
  AAA(1,:)=AAA(1,:)-AAA(t,:);
end
x0=simplex1(AAA,p);
A0=zeros(1,p);
M0=A0;
for i=1:pp
  if A(i,:)*x0==b(i)
      A0=[A0;A(i,:)];
  else
      M0=[M0;A(i,:)];
  end
end
[r,j]=size(A0);
```

```
[ll,kk]=size(M0);
if ll>1
  M00=M0(2:ll,:);
  [k,kk]=size(M00);
end
A00=A0(2:r,:);
[m,n]=size(A00);
b1=ones(m,1);
Y = geninv(A00);
p=Y*b1;
x00=x0;
x0=x0+delta*p;
if ll>1
  while delta>10^-5
     i=0;
     while i<ll-1
        i=i+1;
        if M00(i,:)*x0<b(i)
              delta=.1*delta;
              x0=x00+delta*p;
              break
        end
     end
     if i==ll-1
        break
     end
  end
end
```

B.2 Procedure simplex1

```
function x0=simplex1(A,p)
[m,n]=size(A);
while abs(A(1,n))>10^-3
    b=A(:,n);
    min1=10^5;
    t=0;
    for i=1:p
        if A(1,i)<min1
            min1=A(1,i);
            t=i;
        end
    end
    min2=10^5;
    s=0;
    for i=2:m
        if A(i,t)>0
            if (b(i)/A(i,t))<min2
                min2=(b(i)/A(i,t));
                s=i;
            end
        end
    end
    min3=A(s,t);
    for i=1:n
        A(s,i)=A(s,i)/min3;
    end
    for j=1:m
        if j==s
            m1=0;
        else
```

```
            m1=A(j,t);
         end
        for i=1:n
          A(j,i)=A(j,i)-m1*A(s,i);
        end
    end
end
[m,n]=size(A);
AA=A(2:m,:);
bb=zeros(p,1);
b=AA(:,n);
for i=1:p
   if norm(AA(:,i))==1
        for j=1:m-1
            if AA(j,i)==1
                l=b(j);
                bb(i)=l;
            end
        end
   end
end
x0=bb;
```

B.3 ccsub000333

```
function [d,Z,Y,Q,R,min1,W0,WW0]=ccsub000333(x,G,g1,W0,WW0,Z,Y,Q,R,
delta,ll,min1,BB,x0,counter)

%This function calculate the step d and the least curvature of G
%using the truncated conjugate gradient method
%n the number of variables of the quadratic objective function
%g the gradient of the quadratic objective function
```

```
%G the Hiession of the quadratic objective function
%g1=g1+G*(BB(:,counter)-x0);
g1=g1+G*(x-x0);
n=length(G);
M=zeros(n);
for i=1:n
    M(i,i)=G(i,i);
end
M=eye(n);
alpha=10^-4;
[m1,n1]=size(W0);
if ll==0
    d=zeros(n,1);
    gz=g1;
    rz=M*g1;
    s=-g1;
    H=G;
    sigma=rz'*gz;
 else
    W01=W0(:,1:n);
    [m0,n0]=size(W01);
    d=zeros(n0-m0,1);
    gz=Z'*g1;
    rz=inv(Z'*Z)*gz;
    s=-rz;
    sigma=rz'*gz;
    H=Z'*G*Z;
end
j=1;
while norm(s)>10^-10
    if s'*H*s<=0
        k=((d'*s))^2+norm(s)^2*(delta^2-norm(d)^2);
```

```
        bb=-d'*s;
        alphatrust=bb+k^.5;
        alphatrust=alphatrust/norm(s)^2;
        d=d+alphatrust*s;
        alpha=alphatrust;
        break
    else
       alpha=sigma/(s'*H*s);
       if norm(d+alpha*s)>=delta
           k=((d'*s))^2+norm(s)^2*(delta^2-norm(d)^2);
           bb=-d'*s;
           sigmaa=bb+k^.5;
           sigmaa=sigmaa/norm(s)^2;
           d=d+sigmaa*s;
           break
       else
           dd=sigma;
           d=d+alpha*s;
           gz=gz+alpha*H*s;
           if ll==0
               rz=M*gz;
           else
               rz=inv(Z'*Z)*gz;
           end
           sigma=gz'*rz;
           beta=sigma/dd;
           s=-rz+beta*s;
       end
     end
    if j==n
        break
```

```
    end
    j=j+1;
end
if ll==0
    d=d;
    ZZZ=g1;
else
    d=Z*d;

    ZZZ=Z'*g1;
    ZZ=Z;
end
rr=length(d);
p=0;
[k1,kk]=size(WW0);
if min(k1,kk)~=0
  WW11=WW0(:,1:n);
  min11=10^20;
  mmm=0;
  for i=1:k1
    if (WW11(i,:)*d<0)&((WW0(i,kk)-WW11(i,:)*x)~=0)
            min111=(WW0(i,kk)-WW11(i,:)*x)/(WW11(i,:)*d);
            if min111<min11
                 min11=min111;
                 p=i;
            end
    end
  end
  alphac=min11;
else
   alphac=10^10;
end
```

```
if alphac<alpha
    d=alphac*d;
end
if (norm(d)<10^-10)
    [m1,n1]=size(W0);
    if min(m1,n1)~=0
         W01=W0(:,1:n1-1);
         [m0,n0]=size(W01);
         [Q,R]=qr(W01');
         R=R(1:m0,:);
         Q1=Q(:,1:m0);
         Q2=Q(:,m0+1:n0);
         Y=Q1*(inv(R'));
         Z=Q2;
         lamda=Y'*g1;
         [pp,ppp]=size(lamda);
         min1=10^5;
         ss=1;
         for i=1:pp
            if lamda(i)<min1
                min1=lamda(i);
                ss=i;
            end

         end
         if min1<0
           if ss==1
                f0f=W0(ss,:);
                W01=W01(2:m0,:);
                W0=W0(2:m1,:);
                WW0=[WW0;f0f];
           else
```

```
                if ss==m0
                  f0f=W0(ss,:);
                  W01=W01(1:m0-1,:);
                  W0=W0(1:m1-1,:);
                  WW0=[WW0;f0f];
                else
                  f0f=W0(ss,:);
                  WWW=W01(1:ss-1,:);
                  WWWW=W01(ss+1:m0,:);
                  WWW1=W0(1:ss-1,:);
                  WWWW1=W0(ss+1:m0,:);
                  W01=[WWW;WWWW];
                  W0=[WWW1;WWWW1];
                  WW0=[WW0;f0f];
                end
            end
            [m0,n0]=size(W01);
            [Q,R]=qr(W01');
            R=R(1:m0,:);
            Q1=Q(:,1:m0);
            Q2=Q(:,m0+1:n0);
            Y=Q1*(inv(R'));
            Z=Q2;
        end
    else
        min1=0;
    end
else
    if alphac<alpha
       [k,kk]=size(WW0);
       RR=WW0;
       if p==1
```

```
    WW0=WW0(2:k,:);
else
    if p==k
        WW0=WW0(1:p-1,:);
    else
        WOW=WW0(1:p-1,:);
        WMW=WW0(p+1:k,:);
        WW0=[WOW;WMW];
    end
end
xx=RR(p,1:n1-1);
[o,oo]=size(W0);
xx=xx';
W0=[W0;RR(p,:)];
[m1,n1]=size(W0);
W01=W0(:,1:n);
[m0,n0]=size(W01);
if o==0
   [Q,R]=qr(W01');
   R1=R(1:m0,:);
   Q1=Q(:,1:m0);
   Q2=Q(:,m0+1:n0);
   Y=Q1*(inv(R1'));
   Z=Q2;
else
    [Q,R]=qr(W01');
    R1=R(1:m0,:);
    Q1=Q(:,1:m0);
    Q2=Q(:,m0+1:n0);
    Y=Q1*(inv(R1'));
    Z=Q2;
 end
```

```
    end
end
```

This procedure selects the index t of a point that will be removed to make a room to the point $x_{opt} + p$, where p minmine the trust region subproblem

B.3.1 calculat02

```
function t=calcul_t02(ww,vv,d,delta,x0,ro,BB,ZZZ,counter)

n=length(d);
xx=BB(:,counter)+d;
B11=zeros(3*n+2,1);
B11(counter,1)=1;
Hw=ZZZ*(ww-vv)+B11;
wHw=(ww-vv)'*ZZZ*(ww-vv)+2*ww(counter)-vv(counter);
B11(counter,1)=0;
mm=(.5*(norm(xx-x0)^4)-wHw);
if counter==1
    B11(2,1)=1;
    ma=ZZZ(2,2)*mm+(B11'*Hw)^2;
    ma1=norm(BB(:,2)-BB(:,counter))^2/delta^2;
    if ma1<1
        ma1=1;
    end
    t=2;
    B11(2,1)=0;
    max1=ma*ma1;
    for i=3:2*n+1
       B11(i,1)=1;
       segma1(i)=ZZZ(i,i)*mm+(B11'*Hw)^2;
```

```
        segma2(i)=norm(BB(:,i)-BB(:,counter))^2/delta^2;
        if segma2(i)<1
            segma2(i)=1;
        end
        if segma1(i)*segma2(i)>max1
                  max1=segma1(i)*segma2(i);
                  t=i;
        end
        B11(i,1)=0;
    end


else
    if counter==2*n+1
        B11(1,1)=1;
        ma=ZZZ(1,1)*mm+(B11'*Hw)^2;
        ma1=norm(BB(:,1)-BB(:,counter))^2/delta^2;
        if ma1<1
             ma1=1;
        end
        t=1;
        max1=ma*ma1;
        B11(1,1)=0;
        for i=2:2*n
             B11(i,1)=1;
             segma1(i)=ZZZ(i,i)*mm+(B11'*Hw)^2;
             segma2(i)=norm(BB(:,i)-BB(:,counter))^2/delta^2;
             if segma2(i)<1
                segma2(i)=1;
             end
             if segma1(i)*segma2(i)>max1
                 max1=segma1(i)*segma2(i);
                 t=i;
```

```
            end
            B11(i,1)=0;
        end
    else
        B11(1,1)=1;
        ma=ZZZ(1,1)*mm+(B11'*Hw)^2;
        ma1=norm(BB(:,1)-BB(:,counter))^2/delta^2;
        if ma1<1
            ma1=1;
        end
        t=1;
        max1=ma*ma1;
        B11(1,1)=0;
        for i=2:counter-1
            B11(i,1)=1;
             segma1(i)=ZZZ(i,i)*mm+(B11'*Hw)^2;
             segma2(i)=norm(BB(:,i)-BB(:,counter))^2/delta^2;
             if segma2(i)<1
                segma2(i)=1;
             end
             if segma1(i)*segma2(i)>max1
                 max1=segma1(i)*segma2(i);
                 t=i;
             end
             B11(i,1)=0;
        end
        B11(counter+1,1)=1;
        ma0=ZZZ(counter+1,counter+1)*mm+(B11'*Hw)^2;
        B11(counter+1,1)=0;
        ma10=norm(BB(:,counter+1)-BB(:,counter))^2/delta^2;
        if ma10<1
            ma10=1;
```

```
        end
        tt=counter+1;
        max2=ma0*ma10;
        for i=counter+2:2*n+1
             B11(i,1)=1;
             segma1(i)=ZZZ(i,i)*mm+(B11'*Hw)^2;
             segma2(i)=norm(BB(:,i)-BB(:,counter))^2/delta^2;
             if segma2(i)<1
                segma2(i)=1;
             end
             if abs(segma1(i))*segma2(i)>max2
                 max2=segma1(i)*segma2(i);
                 tt=i;
             end
             B11(i,1)=0;
        end
        if max2>max1
            t=tt;
        end
    end
end
```

B.4 aupdate01

```
function [BB,g,G,ZZZ,Z,x0,counter,lamda]=update01(x0,g,G,t,ww,vv,d,xx,
BB,ZZZ,Z,yy1,counter,pp,ppp,MOM,RATIO,g00,s1)
%This function update01 the inverse matrix H and
%the gradient vector and the Hiessian matrix
%using the least frobenius norm techique
%n the number of variables of the quadratic objective function
%g the gradient of the quadratic objective function
```

```
%G the Hiessain of the quadratic objective function
n=length(d);
E=ones(1,2*n+1);
if norm(d)^2<=10^(-3)*norm(BB(:,counter)-x0)^2
   xxx=x0;
   xav=.5*(x0+BB(:,counter));
   s=BB(:,counter)-x0;
   for i=1:2*n+1
       Y(:,i)=(s'*(BB(:,i)-xav))*(BB(:,i)-xav)+0.25*(norm(s)^2)*s;
   end
   omiga=ZZZ(1:2*n+1,1:2*n+1);
   teta=ZZZ(2*n+3:3*n+2,1:2*n+1);
   meta=ZZZ(2*n+3:3*n+2,2*n+3:3*n+2);
   KK=teta;
   teta=teta+Y*omiga;
   meta=meta+Y*KK'+KK*Y'+Y*omiga*Y';
   RR=ZZZ(2*n+2,1:2*n+1);
   teta1=[RR;teta];
   DD=ZZZ(2*n+2,2*n+3:3*n+2);
   meta1=[DD;meta];
   HH=[omiga teta1'];
   MM=ZZZ(2*n+2:3*n+2,2*n+2);
   meta2=[MM meta1];
   HHH=[teta1 meta2];
   ZZZ=[HH;HHH];
   x0=BB(:,counter);
   for i=1:2*n+1
                  v(i)=.5*((BB(:,i)-x0)'*(BB(:,counter)-x0))^2;
   end
              v1=v';
              wm=1;
              v2=(BB(:,counter)-x0);
```

```
              vv=[v1;wm;v2];
              for i=1:2*n+1
                   w(i)=.5*((BB(:,i)-x0)'*(xx-x0))^2;
              end
              w1=w';
              w2=(xx-x0);
              ww=[w1;wm;w2];
   g=g+G*s;
end
BB(:,t)=xx;
[g,G,ZZZ,Z,segma,tau,sol1,lamda,yy1]=aupdate0(g,G,t,ww,vv,xx,x0,n,BB
,ZZZ,Z,yy1,counter,pp,ppp,MOM,RATIO,g00,s1);
if ppp<pp
    counter=t;
end
```

B.4.1 aupdate0

```
function [g,G,ZZZ,Z,segma,tau,sol1,lamda,yy1]=aupdate0(g,G,t,ww,vv,
xx,x0,n,BB,ZZZ,Z,yy1,counter,pp,ppp,MOM,RATIO,g00,s1)
%This function update the inverse matrix H and the gradient
%vector and the Hiessian matrix
%using the least frobenius norm techique
%n the number of variables of the quadratic objective function
%g the gradient of the quadratic objective function
%G the Hiessain of the quadratic objective function
n=length(x0);
x1=ones(1,3);
ZZ=zeros(2*n+1);
d=xx-BB(:,counter);
gg=g+G*(BB(:,counter)-x0);
B11=zeros(3*n+2,1);
```

```
B1=zeros(3*n+2,1);
B1(t,1)=1;
B11(counter,1)=1;
alpha=ZZZ(t,t);
Hw=ZZZ*(ww-vv)+B11;
wHw=(ww-vv)'*ZZZ*(ww-vv)+2*ww'*B11-vv'*B11;
beta=.5*(norm(xx-x0))^4-wHw;
tau1=Hw;
tau=tau1(t);
segma=alpha*beta+tau^2;
chop1=B1-Hw;
chop=chop1(1:2*n+1,1);
HH1=alpha*(B1-Hw)*(B1-Hw)';
HH3=-beta*ZZZ*B1*B1'*ZZZ;
HH2=tau*((ZZZ*B1*(B1-Hw)'+(B1-Hw)*B1'*ZZZ));
ZZZ=ZZZ+segma^(-1)*(HH1+HH2+HH3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[Z,mm,mm0,mm00]=updateomiga0(Z,yy1,t);
if mm==n
    if mm0~=0
       Z(:,mm0)=abs(segma)^(-.5)*(tau*Z(:,mm0)+Z(t,mm0)*chop);
       yy1(1,mm0)=sign(segma)*yy1(1,mm0);
    end
else
    if beta>0
          eczi=tau^2+beta*(Z(t,mm0))^2;
          Z(:,mm0)=abs(eczi)^(-.5)*(tau*Z(:,mm0)+Z(t,mm0)*chop);
          Z(:,mm00)=abs((eczi*segma))^(-.5)*(-beta*Z(t,mm0)*Z(t,mm00)*
          Z(:,mm0)+eczi*Z(:,mm00)+tau*Z(t,mm00)*chop);
          yy1(1,mm0)=1;
          yy1(1,mm00)=-sign(segma);
    else
```

```
          ecsi=tau^2-beta*Z(t,mm00)^2;
          Z(:,mm0)=abs(ecsi*segma)^-.5*(ecsi*Z(:,mm0)+beta*Z(t,mm0)*
          Z(t,mm00)*Z(:,mm00)+tau*Z(t,mm0)*chop);
          Z(:,mm00)=abs(ecsi)^-.5*(tau*Z(:,mm00)+Z(t,mm00)*chop);
          yy1(1,mm0)=sign(segma)*yy1(1,mm0);
          yy1(1,mm00)=-1;
    end


end
yyy=eye(n);
for i=1:n
   yyy(:,i)=yy1(i)*yyy(:,i);
end
for i=1:n
  ZZ=ZZ+yy1(i)*Z(:,i)*Z(:,i)';
end
VV=ZZZ(2*n+2:3*n+2,1:2*n+1);
VV1=ZZZ(2*n+2:3*n+2,2*n+2:3*n+2);
ZZZ=[ZZ VV';VV VV1];
s1=s1+1;
x1(1,mod(s1,3)+1)=RATIO;
min0=x1(1);
for i=2:3
  if x1(i)>min0
      min0=x1(i);
  end
end
xxx=zeros(n);
if min0<=.01
   for i=1:2*n+1
       r(i)=MOM(i)-MOM(counter);
   end
```

```
   rr=zeros(n+1,1);
   r=r';
   r=[r;rr];
   sol3=r'*ZZZ;
   sol3=sol3';
   alamda=sol3(1:2*n+1);
   sol4=sol3(2*n+3:3*n+2);
   xxxx=xxx;
end
sol1=((ppp-pp)-d'*gg-.5*d'*G*d)*ZZZ*B1;
lamda=sol1(1:2*n+1);
sol2=sol1(2*n+3:3*n+2);
for k=1:2*n+1
   xxx=xxx+lamda(k)*(BB(:,k)-x0)*(BB(:,k)-x0)';
end
if min0<=.01
   if (norm(sol4)<=.1*norm(g00))
       for k=1:2*n+1
           xxxx=xxxx+alamda(k)*(BB(:,k)-x0)*(BB(:,k)-x0)';
       end
       g=sol4;
       G=xxxx;
   else
      G=G+xxx;
      g=g+sol2;
   end
else
    G=G+xxx;
    g=g+sol2;
end
```

This procedure tries to improve the geomegtry of the interpolation set

B.5 RESCUE11

```
function [g,G,BB,ZZZ,counter,fcounter0,MOM,x0]=RESCUE11(A,b,BB,
counter,ZZZ,delta,MOM,g,G,x0)
epo=delta;
[m,n]=size(A);
fcounter0=2*n+1;
p=n;
pp=m;
xx=x0;
x0=BB(:,counter);
A0=zeros(1,n);
M0=A0;
for i=1:pp
  if A(i,:)*x0==b(i)
      A0=[A0;A(i,:)];
  else
      M0=[M0;A(i,:)];
  end
end
[r,j]=size(A0);
[ll,kk]=size(M0);
if ll>1
  M00=M0(2:ll,:);
  [k,kk]=size(M00);
end
A00=A0(2:r,:);
[m,n]=size(A00);
fcounter0=0;
b1=ones(m,1);
Y = geninv(A00);
```

```
p=Y*b1;
x00=x0;
x0=x0+delta*p;
if ll>1
  while delta>10^-5
     i=0;
     while i<ll-1
        i=i+1;
        if M00(i,:)*x0<b(i)
              delta=.1*delta;
              x0=x00+delta*p;
              break
        end
     end
     if i==ll-1
        break
     end
  end
end
n=length(x0);
H=eye(n);
HH=zeros(n);
RR=zeros(n+1);
MM=zeros(n);
yy1=ones(1,n);
s=zeros(1,n);
rr=eye(n+1);
rrr=rr(:,1);
first=ones(1,n);
first=-2^(.5)*(epo)^(-2)*first;
for i=1:n
   HH(:,i)=x0+epo*H(:,i);
```

```
   M(:,i)=x0-epo*H(:,i);
   HHHH(:,i)=.5*(2)^.5*(epo)^(-2)*H(:,i);
end
BB=[x0 HH M];
kk=(2*epo)^-1*eye(n);
kkk=[s;kk];
kk1=-(2*epo)^-1*eye(n);
kkk1=[s;kk1];
tr=[kkk kkk1];
BBB=[rrr tr];
TT=BBB';
RR=[BBB RR];
Z=[first;HHHH;HHHH];
yyy=eye(n);
for i=1:n
   yyy(:,i)=yy1(i)*yyy(:,i);
end
ZZ=Z*yyy*Z';
ZZZ=[ZZ TT];
ZZZ=[ZZZ;RR];
counter=1;
MOM(1)=F1(BB(:,1));
minval=MOM(1);
for i=2:2*n+1
    MOM(i)=F1(BB(:,i));
   if MOM(i)<minval
      minval=MOM(i);
      counter=i;
   end
end
G1=ones(2*n,1);
G2=eye(n);
```

```
for j=2:2*n+1
   for i=1:n
      M(j-1,i)=BB(i,j)-x0(i);
      M1(j-1,i)=(BB(i,j)-x0(i))^2;
   end
end
M1=.5*M1;
MM=[M M1];
c=MOM(1);
for j=2:2*n+1
   s(j-1)=MOM(j)-c;
end
s1=s';
inv(MM);
xx=inv(MM)*s1;
g=xx(1:n);
xxx=xx(n+1:2*n);
for j=1:n
   G(:,j)=xxx(j)*G2(:,j);
end
```

This function calculate the step d when d from sup3 is rejected using the extension of truncated conjugate gradient method n the number of variables of the quadratic objective function

B.6 bbbBIGLAG01

```
function [d,delta]=bbbBIGLAG01(t,g,G,BB,ZZZ,x0,delta1,ro1,dist1,counter)

xx=BB(:,counter);
x1=xx;
n=length(g);
```

```
GG=zeros(n);
delta=delta1;
sol=ZZZ(:,t);
lamda1=sol(1:2*n+1);
g1=sol(2*n+3:3*n+2);
for i=1:2*n+1
   GG=GG+lamda1(i)*(BB(:,i)-x0)*(BB(:,i)-x0)';
end
g=g1+GG*(BB(:,counter)-x0);
mmax=0;
if counter==1
    for j=2:2*n+1
       alphaa(j)=mmoh(j,counter,BB,g,GG,delta);
    end
else
    if counter==2*n+1
       for j=1:2*n
           alphaa(j)=mmoh(j,counter,BB,g,GG,delta);
       end
    else
        for j=1:counter-1
            alphaa(j)=mmoh(j,counter,BB,g,GG,delta);
        end
        for j=counter+1:2*n+1
            alphaa(j)=mmoh(j,counter,BB,g,GG,delta);
        end
    end
end
if counter==1
    xxx=BB(:,2);
    rr=alphaa(2)*(BB(:,2)-BB(:,counter));
    if norm(rr)>delta
```

```
        alphaa(2)=delta/(norm(BB(:,2)-BB(:,counter)));
    end
    abc(2)=abs(alphaa(2)*(BB(:,2)-BB(:,counter))'*g+.5*
    alphaa(2)^2*(BB(:,2)-BB(:,counter))'*GG*(BB(:,2)-BB(:,counter)));
    aabc(2)=abc(2)^2*(.5*ZZZ(2,2)*alphaa(2)^2*(1-alphaa(2))^2*
    norm(BB(:,2)-BB(:,counter))^4+abc(2)^2);
    l=2;
    max0=aabc(2);
    for j=3:2*n+1
        xxx=BB(:,j);
        rr=alphaa(j)*(BB(:,j)-BB(:,counter));
        if norm(rr)>delta
            alphaa(j)=delta/(norm(BB(:,j)-BB(:,counter)));
        end
        abc(j)=abs(alphaa(j)*(BB(:,j)-BB(:,counter))'
        *g+.5*alphaa(j)^2*(BB(:,j)-BB(:,counter))'*GG*
        (BB(:,j)-BB(:,counter)));
        aabc(j)=abc(j)^2*(.5*ZZZ(t,t)*alphaa(j)^2*
        (1-alphaa(j))^2*norm(BB(:,j)-BB(:,counter))^4+abc(j)^2);
        if aabc(j)>max0
           max0=aabc(j);
           l=j;
        end
    end
else
    if counter==2*n+1
        xxx=BB(:,1);
        rr=alphaa(1)*(BB(:,1)-BB(:,counter));
        if norm(rr)>delta
            alphaa(1)=delta/(norm(BB(:,1)-BB(:,counter)));
        end
        abc(1)=abs(alphaa(1)*(BB(:,1)-BB(:,counter))'*
```

```
        g+.5*alphaa(1)^2*(BB(:,1)-BB(:,counter))'*GG*
        (BB(:,1)-BB(:,counter)));aabc(1)=abc(1)^2*(.5*ZZZ(1,1)*alphaa(1)^2*
        (1-alphaa(1))^2*norm(BB(:,1)-BB(:,counter))^4+abc(1)^2);
        l=1;
        max0=aabc(1);
        for j=2:2*n
             xxx=BB(:,j);
             rr=alphaa(j)*(BB(:,j)-BB(:,counter));
             if norm(rr)>delta
                 alphaa(j)=delta/(norm(BB(:,j)-BB(:,counter)));
             end
             abc(j)=abs(alphaa(j)*(BB(:,j)-BB(:,counter))'*g+.5*
             alphaa(j)^2*(BB(:,j)-BB(:,counter))'*GG*(BB(:,j)-BB(:,counter)));
             aabc(j)=abc(j)^2*(.5*ZZZ(t,t)*alphaa(j)^2*(1-alphaa(j))^2*
             norm(BB(:,j)-BB(:,counter))^4+abc(j)^2);
             if aabc(j)>max0
                max0=aabc(j);
                l=j;
             end
       end
    else
        xxx=BB(:,1);
        rr=alphaa(1)*(BB(:,1)-BB(:,counter));
        if norm(rr)>delta
            alphaa(1)=delta/(norm(BB(:,1)-BB(:,counter)));
        end
        abc(1)=abs(alphaa(1)*(BB(:,1)-BB(:,counter))'*g+.5*
        alphaa(1)^2*(BB(:,1)-BB(:,counter))'*GG*(BB(:,1)-BB(:,counter)));
        aabc(1)=abc(1)^2*(.5*ZZZ(1,1)*alphaa(1)^2*(1-alphaa(1))^2*
        norm(BB(:,1)-BB(:,counter))^4+abc(1)^2);
        l=1;
        max0=aabc(1);
```

```
for j=2:counter-1
   xxx=BB(:,j);
   rr=alphaa(j)*(BB(:,j)-BB(:,counter));
   if norm(rr)>delta
       alphaa(j)=delta/(norm(BB(:,j)-BB(:,counter)));
   end
   abc(j)=abs(alphaa(j)*(BB(:,j)-BB(:,counter))'*g+.5*alphaa(j)^2*(BB(:,j)-
   BB(:,counter))'*GG*(BB(:,j)-BB(:,counter)));
   aabc(j)=abc(j)^2*(.5*ZZZ(t,t)*alphaa(j)^2*(1-alphaa(j))^2*
   norm(BB(:,j)-BB(:,counter))^4+abc(j)^2);
   if aabc(j)>max0
       max0=aabc(j);
       l=j;
   end
end
xxx=BB(:,counter+1);
rr=alphaa(counter+1)*(BB(:,counter+1)-BB(:,counter));
if norm(rr)>delta
    alphaa(counter+1)=delta/(norm(BB(:,counter+1)-BB(:,counter)));
end
abc(counter+1)=abs(alphaa(counter+1)*(BB(:,counter+1)-BB(:,counter))'*g+.5*al
*GG*(BB(:,counter+1)-BB(:,counter)));
aabc(counter+1)=abc(counter+1)^2*
(.5*ZZZ(counter+1,counter+1)*alphaa(counter+1)^2*(1-alphaa(counter+1))^2*
norm(BB(:,counter+1)-BB(:,counter))^4+abc(counter+1)^2);
ll=counter+1;
max00=aabc(counter+1);
for j=counter+1:2*n+1
    xxx=BB(:,j);
    rr=alphaa(j)*(BB(:,j)-BB(:,counter));
    if norm(rr)>delta
        alphaa(j)=delta/(norm(BB(:,j)-BB(:,counter)));
```

```
            end
            abc(j)=abs(alphaa(j)*(BB(:,j)-BB(:,counter))'*g+.5*
            alphaa(j)^2*(BB(:,j)-BB(:,counter))'*GG*(BB(:,j)-BB(:,counter)));
            aabc(j)=abc(j)^2*(.5*ZZZ(t,t)*alphaa(j)^2*(1-alphaa(j))^2*
            norm(BB(:,j)-BB(:,counter))^4+abc(j)^2);
            if aabc(j)>max00
               max00=aabc(j);
               ll=j;
            end
        end
        if max00>max0
            l=ll;
        end
    end
end
d=alphaa(l)*(BB(:,l)-BB(:,counter));
```

B.6.1 mmoh

```
function max1=mmoh(j,counter,BB,g,G,delta)
n=length(g);
xx=BB(:,counter);
xxx=BB(:,j);
if ((xxx-xx)'*G*(xxx-xx))>10^-6
     max1=-(xxx-xx)'*g/((xxx-xx)'*G*(xxx-xx));
else
     if ((xxx-xx)'*G*(xxx-xx))<-10^-6
         max1=(xxx-xx)'*g/((xxx-xx)'*G*(xxx-xx));
     else
         tt=0:0.1:1;
         max1=0;
         ttt=1;
```

```
         max1=FFF000(tt(1),j,counter,BB,g,G);
         for i=2:length(tt)
              z(i)=FFF000(tt(i),j,counter,BB,g,G);
              if z(i)>max1
                 max1=z(i);
                 ttt=i;
              end
         end
     end
end
```

This procedure calculate the index of the interpolation that will removed to improve the geometry of the interpolation points

B.6.2 table07

```
function [dist1,MOVE]=table07(BB,n,counter)


m=2*n+1;
if counter==1
  MOVE=2;
  dist1=norm(BB(:,2)-BB(:,counter));
   for i=3:m
     if norm(BB(:,i)-BB(:,counter))>dist1
        dist1=norm(BB(:,i)-BB(:,counter));
        MOVE=i;
     end
   end
else
  if counter==2*n+1
     dist1=norm(BB(:,1)-BB(:,counter));
     MOVE=1;
```

```
    for i=2:counter-1
      if norm(BB(:,i)-BB(:,counter))>dist1
         dist1=norm(BB(:,i)-BB(:,counter));
         MOVE=i;
       end
    end
  else
    dist1=norm(BB(:,1)-BB(:,counter));
    MOVE=1;
    for i=2:counter-1
      if norm(BB(:,i)-BB(:,counter))>dist1
         dist1=norm(BB(:,i)-BB(:,counter));
         MOVE=i;
       end
    end
    move1=counter+1;
    dist2=norm(BB(:,counter+1)-BB(:,counter));
    for i=counter+2:m
      if norm(BB(:,i)-BB(:,counter))>dist1
          dist2=norm(BB(:,i)-BB(:,counter));
          move1=i;
       end
    end
    if dist2>dist1
       MOVE=move1;
       dist1=dist2;
    end
  end
end
```

This procedure solves the simple bound problems

B.7 BOBYQA

```
function [x,fcounter]=BOBYQA(x0,a,b,ro,roend)
%this procedure solves the simple bound problems
st=cputime;
n=length(x0);
pp=0;
ss1=1;
s1=1;
delta=ro;
[g,G,BB,Z,ZZZ,yy1,counter,MOM,x0]=binitial001(x0,ro,a,b);
[g,G]=F2(BB,x0);
g00=g;
fcounter=2*n+1;
e=1;
fcounter1=0;
fcounter2=0;
fcounter3=0;
fcounter5=0;
fcounter6=0;
uu=0;
s1=1;
x1=ro*ones(1,3);
xx1=ro*ones(1,3);
rtr=0;
ppp=0;
while ro>roend
   [mincrv,d]=sub003(x0,BB,g,G,delta,counter,a,b);
   if norm(d)>=.5*ro
   xx=BB(:,counter)+d;
   for i=1:2*n+1
        v(i)=.5*((BB(:,i)-x0)'*(BB(:,counter)-x0))^2;
```

```
end
v1=v';
wm=1;
v2=(BB(:,counter)-x0);
vv=[v1;wm;v2];
for i=1:2*n+1
     w(i)=.5*((BB(:,i)-x0)'*(xx-x0))^2;
end
w1=w';
w2=(xx-x0);
ww=[w1;wm;w2];
B11=zeros(3*n+2,1);
B11(counter,1)=1;
t=calcul_t02(ww,vv,d,delta,x0,ro,BB,ZZZ,counter);
alpha=ZZZ(t,t);
Hw=ZZZ*(ww-vv)+B11;
wHw=(ww-vv)'*ZZZ*(ww-vv)+2*ww'*B11-vv'*B11;
B11(counter,1)=0;
beta=.5*(norm(xx-x0))^4-wHw;
tau1=Hw;
tau=tau1(t);
segma=alpha*beta+tau^2;
if segma<=.5*tau^2
       [g,G,BB,ZZZ,counter,fcounter0,MOM,x0]=RESCUE1(a,b,BB,
       counter,ZZZ,delta,MOM,g,G,x0);
       fcounter=fcounter+fcounter0;
       rtr=1;
else
    pp=MOM(counter);
    ppp=F1(xx);
    RATIO=(pp-ppp)/(-d'*(g+G*(BB(:,counter)-x0))-.5*d'*G*d);
    fcounter=fcounter+1;
```

```
delta=revisedelta(RATIO,d,delta,ro);
[BB,g,G,ZZZ,Z,x0,counter,lamda]=aupdate01(x0,g,G,t,ww,vv,d,xx,BB,ZZZ,Z,
yy1,counter,pp,ppp,MOM,RATIO,g00,ss1);
%[BB,g,G,ZZZ,Z,x0,counter,lamda]=update02(x0,g,G,t,ww,vv,d,xx,BB,ZZZ,Z,
yy1,counter,pp,ppp,MOM,RATIO,g00,s1);
ss1=ss1+1;
if counter==t
    MOM(counter)=ppp;
end
if RATIO<0.1
   [dist1,MOVE]=table07(BB,n,counter);
   if dist1>=2*delta
       [d,delta2]=bBIGLAG01(MOVE,g,G,BB,ZZZ,x0,delta,ro,dist1
       ,counter,a,b);
       ddd=d;
       xx0=BB(:,counter)+d;
       for i=1:2*n+1
             v(i)=.5*((BB(:,i)-x0)'*(BB(:,counter)-x0))^2;
       end
       v1=v';
       wm=1;
       v2=(BB(:,counter)-x0);
       vv=[v1;wm;v2];
       for i=1:2*n+1
            w(i)=.5*((BB(:,i)-x0)'*(xx0-x0))^2;
       end
       w1=w';
       w2=(xx0-x0);
       ww=[w1;wm;w2];
       t=MOVE;
       alpha=ZZZ(t,t);
       B11=zeros(3*n+2,1);
```

```
        B11(counter,1)=1;
        Hw=ZZZ*(ww-vv)+B11;
        wHw=(ww-vv)'*ZZZ*(ww-vv)+2*ww'*B11-vv'*B11;
        beta=.5*(norm(xx0-x0))^4-wHw;
        tau1=Hw;
        B11(counter,1)=0;
        tau=tau1(t);
        segma=alpha*beta+tau^2;
        if segma<=.5*tau^2
               [g,G,BB,ZZZ,counter,fcounter0,MOM,x0]=RESCUE1(a,b,BB,
               counter,ZZZ,delta,MOM,g,G,x0);
               fcounter=fcounter+fcounter0;
        else
            pp=MOM(counter);
            ppp=F1(xx0);
            uu=uu+1;
            fcounter1=fcounter1+1;
            [BB,g,G,ZZZ,Z,x0,counter,lamda]=aupdate01(x0,g,G,t,ww,vv,d,xx0,BB,Z
            yy1,counter,pp,ppp,MOM,RATIO,g00,ss1);
            %[BB,g,G,ZZZ,Z,x0,counter,lamda]=update02(x0,g,G,MOVE,ww,vv,d,xx0,B
            yy1,counter,pp,ppp,MOM,RATIO,g00,s1);
        end
        if counter==MOVE
            MOM(counter)=ppp;
        end
    else
        max00=norm(d);
        if delta>max00;
              max00=delta;
        end
        if (max00<=ro) & RATIO<=0
               ro1=ro;
```

```
                    ro=revisedro(ro,roend);
                    max00=0.5*ro1;
                    if ro>max00
                       max00=ro;
                    end
                    delta=max00;
              end
          end
       end
  end
else
    x1(1,mod(s1,3)+1)=norm(d);
    xx1(1,mod(s1,3)+1)=abs(MOM(counter)-ppp);
    s1=s1+1;
    min0=x1(1);
    min1=xx1(1);
    for i=2:3
        if x1(i)>min0
           min0=x1(i);
        end
        if xx1(i)>min1
           min1=xx1(i);
        end
    end
    if( min0<=ro)&(min1<=ro^2*mincrv/8)
        ro1=ro;
        ro=revisedro(ro,roend);
        max1=.5*ro1;
        if ro>max1
            max1=ro;
        end
        delta=max1;
```

```
else
    ro1=ro;
    ro=revisedro(ro,roend);
    max1=.5*ro1;
    if ro>max1
         max1=ro;
    end
    delta=max1;
    RATIO=-1;
    [dist,MOVE]=table07(BB,n,counter);
    if dist>=2*delta
          [d,delta2]=bBIGLAG01(MOVE,g,G,BB,ZZZ,x0,delta,ro,
          dist,counter,a,b);
          xx00=BB(:,counter)+d;
          for i=1:2*n+1
               v(i)=.5*((BB(:,i)-x0)'*(BB(:,counter)-x0))^2;
          end
          v1=v';
          wm=1;
          v2=(BB(:,counter)-x0);
          vv=[v1;wm;v2];
          for i=1:2*n+1
                 w(i)=.5*((BB(:,i)-x0)'*(xx00-x0))^2;
          end
          w1=w';
          w2=(xx00-x0);
          ww=[w1;wm;w2];
          t=MOVE;
          B11=zeros(3*n+2,1);
          B11(counter,1)=1;
          alpha=ZZZ(t,t);
          Hw=ZZZ*(ww-vv)+B11;
```

```
    wHw=(ww-vv)'*ZZZ*(ww-vv)+2*ww'*B11-vv'*B11;
    B11(counter,1)=0;
    beta=.5*(norm(xx00-x0))^4-wHw;
    tau1=Hw;
    B11(counter,1)=0;
    tau=tau1(t);
    segma=alpha*beta+tau^2;
    if segma<=.5*tau^2
           [g,G,BB,ZZZ,counter,fcounter0,MOM,x0]=RESCUE1(a,b,BB,
           counter,ZZZ,delta,MOM,g,G,x0);
           fcounter=fcounter+fcounter0;
    else
       pp=MOM(counter);
       ppp=F1(xx00);
       uu=uu+1;
       fcounter5=fcounter5+1;
       [BB,g,G,ZZZ,Z,x0,counter,lamda]=aupdate01(x0,g,G,t,ww,vv,d,xx00,
       BB,ZZZ,Z,yy1,counter,pp,ppp,MOM,RATIO,g00,ss1);
       %[BB,g,G,ZZZ,Z,x0,counter,lamda]=update02(x0,g,G,MOV
       %E,ww,vv,d,xx00,BB,ZZZ,Z,yy1,counter,pp,ppp,MOM,RATIO,g00,s1);
    end
    if counter==MOVE
       MOM(counter)=ppp;
    end
else
    max1=norm(d);

    if delta>max1;
       max1=delta;
    end
    if (max1<=ro)&(RATIO<=0)
         ro1=ro;
```

```
                    ro=revisedro(ro,roend);
                    max1=.5*ro1;
                    if ro>max1
                       max1=ro;
                    end
                    delta=max1;
                end
            end
         end
 end
 e=e+1;
  if e==50000
      break
  end

end
fcounter=fcounter+fcounter1+fcounter5;
if norm(d)<.5*ro
  if ppp<pp;
      BB(:,counter)=BB(:,counter)+d;
  end
end
yy2=ones(n,1);
x=BB(:,counter);
eee=norm((yy2-x),inf);
ro;
et=cputime;
t=et-st;
```

B.7.1 binitial001

```
function [g,G,BB,Z,ZZZ,yy1,counter,MOM,x0]=binitial001(x0,epo,a,b)
```

```
% in this function we initialize the m=2*n+1 points
%also we initialize the first invers matrix H
%the initial point
%x0=initialpoint(n);
n=length(x0);
delta=epo;
H=eye(n);
M=zeros(n);
HH=zeros(n);
RR=zeros(n+1);
HHH=zeros(n);
MM=zeros(n);
yy1=ones(1,n);
rr=eye(n+1);
rrr=rr(:,1);
M0=zeros(n,2*n+1);
N0=M0';
e=ones(2*n+1);
for i=1:n
    if x0(i)<a(i)
        x0(i)=a(i);

    else
        if x0(i)>b(i)
            x0(i)=b(i);
        end
    end
end
for i=1:n
    if (a(i)<x0(i))&(x0(i)<a(i)+epo)
       x0(i)=a(i)+epo ;
    end
```

```
    if ((b(i)-epo)<x0(i))&(x0(i)<b(i))
        x0(i)=b(i)-epo;
    end
end
for i=1:n
    if (a(i)< x0(i)) &( x0(i)<b(i))
         HH(:,i)=x0+epo*H(:,i);
         M(:,i)=x0-epo*H(:,i);
         alpha(i)=epo;
         beta(i)=-epo;
    else
        if x0(i)==a(i)
          HH(:,i)=x0+epo*H(:,i);
          M(:,i)=x0+2*epo*H(:,i);
          alpha(i)=epo;
          beta(i)=2*epo;
        else
           HH(:,i)=x0-epo*H(:,i);
           M(:,i)=x0-2*epo*H(:,i);
           alpha(i)=-epo;
           beta(i)=-2*epo;
        end
    end
    %HHHH(:,i)=.5*(2)^.5*(epo)^(-2)*H(:,i);
end
BB=[x0 HH M];
M00=zeros(1,2*n);
M00=[1 M00];
for i=1:n
    M0(i,1)=-1/alpha(i)-1/beta(i);
    M0(i,i+1)=beta(i)/(alpha(i)*(beta(i)-alpha(i)));
    M0(i,n+i+1)=alpha(i)/(beta(i)*(alpha(i)-beta(i)));
```

```
    N0(1,i)=2^.5/(alpha(i)*beta(i));
    N0(i+1,i)=2^.5/(alpha(i)*(beta(i)-alpha(i)));
    N0(n+i+1,i)=2^.5/(beta(i)*(alpha(i)-beta(i)));
end
M0=[M00;M0];
Z=N0;
RR=[M0 RR];
ZZ=N0*N0';
ZZZ=[ZZ M0'];
ZZZ=[ZZZ;RR];
G1=ones(2*n,1);
G2=eye(n);
for j=2:2*n+1
   for i=1:n
      M(j-1,i)=BB(i,j)-x0(i);
      M1(j-1,i)=(BB(i,j)-x0(i))^2;
   end
end
M1=.5*M1;
MM=[M M1];
counter=1;
MOM(1)=F1(BB(:,1));
minval=MOM(1);
for i=2:2*n+1
    MOM(i)=F1(BB(:,i));
   if MOM(i)<minval
      minval=MOM(i);
      counter=i;
   end
end
%%%%
c=MOM(1);
```

```
for j=2:2*n+1
   s(j-1)=MOM(j)-c;
end
s1=s';
xx=inv(MM)*s1;
g=xx(1:n);
xxx=xx(n+1:2*n);
for j=1:n
   G(:,j)=xxx(j)*G2(:,j);
end
```

This procedure updates the trust region radius

B.7.2 revisedelta

```
function delta=revisedelta(RATIO,d,delta,ro)

if RATIO<=.1
   delta=.5*norm(d);
else
  if (.1<RATIO & RATIO<=.7)
      max=norm(d);
      if (.5*delta) > max
         max=.5*delta;
       end
       delta=max;
   else
       max=2*norm(d);
       if .5*delta>max
          max=.5*delta;
       end
       delta=max;
```

```
   end
end
if delta<=1.5*ro
   delta=ro;
end
```

This procedure tries to improve thye geomegtry of the interpolation set of BOBYQA

B.7.3 RESCUE1

```
function [g,G,BB,ZZZ,counter,fcounter0,NON0,x0]=RESCUE1(a,b,BB,c
ounter,ZZZ,delta,MOM,g,G,x0)

fcounter0=0;
xx=x0;
x0=BB(:,counter);
g=g+G*(BB(:,counter)-xx);
xx=x0;
gg=g;
GG=G;
n=length(x0);
H=eye(n);
epo=delta;
e=ones(n,1);
RR=zeros(n+1);
yy1=ones(1,n);
B1=zeros(3*n+2,1);
v=B1;
v(2*n+1,1)=1;
yy1=ones(1,n);
NON=zeros(2*n+1,1);
NON0=NON;
```

```
rr=eye(n+1);
rrr=rr(:,1);
M0=zeros(n,2*n+1);
N0=M0';
NONN=NON;
B11=zeros(3*n+2,1);
B11(counter,1)=1;
if (a<=(xx+delta*e))&(xx+delta*e<=b)
    alpha=delta*e;
    beta=-delta*e;
    for i=1:n
        BBB(:,i)=xx+delta*H(:,i);
        BBBB(:,i)=xx-delta*H(:,i);
    end
else
    for i=1:n
        if xx(i)+delta<b(i)
            alpha(i)=delta;
            BBB(:,i)=xx+delta*H(:,i);
        else
            alpha(i)=-delta;
            BBB(:,i)=xx-delta*H(:,i);
        end
    end
    for i=1:n
        if alpha(i)>0
            beta(i)=(a(i)-xx(i));
        else
            beta(i)=(b(i)-xx(i));
        end
    end
    for i=1:n
```

```
        if abs(beta(i))<.5*delta
            beta(i)=.5*delta;
        end
    end
    for i=1:n
        BBBB(:,i)=xx+beta(i)*H(:,i);
    end
end
BBB=[x0 BBB BBBB];
M00=zeros(1,2*n);
M00=[1 M00];
for i=1:n
    M0(i,1)=-1/alpha(i)-1/beta(i);
    M0(i,i+1)=beta(i)/(alpha(i)*(beta(i)-alpha(i)));
    M0(i,n+i+1)=alpha(i)/(beta(i)*(alpha(i)-beta(i)));
    N0(1,i)=2^.5/(alpha(i)*beta(i));
    N0(i+1,i)=2^.5/(alpha(i)*(beta(i)-alpha(i)));
    N0(n+i+1,i)=2^.5/(beta(i)*(alpha(i)-beta(i)));
end
M0=[M00;M0];
Z=N0;
RR=[M0 RR];
ZZ=N0*N0';
ZZZ=[ZZ M0'];
ZZZ=[ZZZ;RR];
G2=eye(n);
for j=2:2*n+1
   for i=1:n
      M(j-1,i)=BBB(i,j)-x0(i);
      M1(j-1,i)=(BBB(i,j)-x0(i))^2;
   end
end
```

```
M1=.5*M1;
MM=[M M1];
counter=1;
MOM(1)=F1(BBB(:,1));
minval=MOM(1);
for i=2:2*n+1
    MOM(i)=F1(BBB(:,i));
   if MOM(i)<minval
      minval=MOM(i);
      counter=i;
   end
end
c=MOM(1);
for j=2:2*n+1
   s(j-1)=MOM(j)-c;
end
s1=s';
xx=inv(MM)*s1;
g=xx(1:n);
xxx=xx(n+1:2*n);
for j=1:n
   G(:,j)=xxx(j)*G2(:,j);
end
NON0=MOM;
fcounter0=2*n+1;
BB=BBB;
g=g+G*(BB(:,counter)-x0);
x0=BB(:,counter);%
```

This procedure update the ro

B.7.4 revisedro

```
function ro=revisedro(ro,ro_end)


if ro<=16*ro_end
         ro=ro_end;
else
    if (16*ro_end<ro) & (ro<=250*ro_end)
         ro=(ro*ro_end)^.5;
    else
       ro=.1*ro;
    end
end
```

This is the main procedure of our software. It compute the minimum of the nonlinear function f(x), subject to the linear constraints Ax=b. The user of LCOBYQA must provide the initial point x0, the coefficient matrix A, the corresponding right hand sideb, the initial trust region ro and the final trust region roend.

B.8 LCOBYQA

```
function [x,fcounter]=LCOBYQA(A,b,x0,ro,roend)


pp=0;
sst=cputime;
[m,n]=size(A);
if mod(m,2)==0
    mr=m/2;
    AA=A(1:mr,:);
    AAA=A(mr+1:m,:);
    if AA==-AAA
        a=b(1:mr);
```

```
        b=-b(mr+1:m);
        [x,fcounter]=BOBYQA(x0,a,b,ro,roend);
    else
        non=length(b);
        mincrv=0;
        min101=-10^10;
        ll=1;
        ss1=1;
        s1=1;
        delta=ro;
        rtt=0;
        k1=1;
        k2=1;
        for i=1:m
          if A(i,:)*x0==b(i)
             k1=k1+1;
          else
             if A(i,:)*x0>b(i)+ro
                 k2=k2+1;
             end
          end
        end
        if k2<m
          if k1<m
             x0=phase1(A,b);
          end
        end
        [g,G,BB,Z,ZZZ,yy1,counter,MOM]=initial01(x0,ro);
        xx=BB(:,counter);
        g00=g;
        g1=g;
        b0=[];
```

```
b1=[];
W=[];
WW=[];
ms=0;
for i=1:m
  if A(i,:)*xx==b(i);
         W=[W;A(i,:)];
         b0=[b0;b(i)];
         ms=i;
  else
         WW=[WW;A(i,:)];
         b1=[b1;b(i)];
  end
end
[o,oo]=size(W);
WW0=[WW b1];
W0=[W b0];
if o==0
    ll=0;
end
if ll==0
    Z0Z=[];
    Y=[];
    Q=[];
    R=[];
else
    [mm,nn]=size(WW0);
    [m1,n1]=size(W0);
    W01=W0(:,1:n1-1);
    [m0,n0]=size(W01);
    [Q,R]=qr(W01');
    R1=R(1:m0,:);
```

```
Q1=Q(:,1:m0);
Q2=Q(:,m0+1:n0);
Y=Q1*(inv(R1'));
Z0Z=Q2;
[l,l1]=size(Z0Z'*g);
if l==0
     lamda=Y'*g;
     [p,pp]=size(lamda);
     min1=10^10;
     ss=0;
     for i=1:p
        if lamda(i)<min1
             min1=lamda(i);
        end
        ss=i;
     end
     if ss==1
           f0f=W0(ss,:);
           W01=W01(2:m0,:);
           W0=W0(2:m1,:);
           WW0=[WW0;f0f];
     else
           if ss==m0
                 f0f=W0(ss,:);
                 W01=W01(1:m0-1,:);
                 W0=W0(1:m1-1,:);
                 WW0=[WW0;f0f];
           else
                 f0f=W0(ss,:);
                 WWW=W01(1:ss-1,:);
                 WWWW=W01(ss+1:m0);
                 WWW1=W0(1:ss-1,:);
```

```
                    WWWW1=W0(ss+1:m0);
                    W01=[WWW;WWWW];
                    W0=[WWW1;WWWW1];
                    WW0=[WW0;f0f];
                end
            end
            [m0,n0]=size(W01);
            [QQ,RR]=qrdelete(Q,R,ss,'col');
            R1=RR(1:m0,:);
            Q1=QQ(:,1:m0);
            Q2=QQ(:,m0+1:n0);
            Y=Q1*(inv(R1'));
            R=RR;
            Q=QQ;
            Z0Z=Q2;
        end
    end
    fcounter=2*n+1;
    e=1;
    fcounter1=0;
    fcounter2=0;
    fcounter3=0;
    fcounter5=0;
    fcounter6=0;
    uu=0;
    s1=1;
    x1=ro*ones(1,3);
    xx1=ro*ones(1,3);
    rtr=0;
    ppp=0;
    RATIO=.2;
    d=zeros(n,1);
```

```
while ro>roend
       d0d=d;
       [d,Z0Z,Y,Q,R,min101,W0,WW0]=ccsub000333(xx,G,g,W0,WW0,Z0Z
       ,Y,Q,R,delta,ll,min101,BB,x0,counter);
       st=0;
       if min101>=0
            rtt=1;
            break
       end
       dd=d;
       [ll,lll]=size(Z0Z);
       if norm(d)>=.5*ro
          xx=xx+d;
          for i=1:2*n+1
              v(i)=.5*((BB(:,i)-x0)'*(BB(:,counter)-x0))^2;
          end
          v1=v';
          wm=1;
          v2=(BB(:,counter)-x0);
          vv=[v1;wm;v2];
          for i=1:2*n+1
               w(i)=.5*((BB(:,i)-x0)'*(xx-x0))^2;
          end
          w1=w';
          w2=(xx-x0);
          ww=[w1;wm;w2];
          B11=zeros(3*n+2,1);
          B11(counter,1)=1;
          t=calcul_t02(ww,vv,d,delta,x0,ro,BB,ZZZ,counter);
          tt=t;
          alpha=ZZZ(t,t);
          Hw=ZZZ*(ww-vv)+B11;
```

```
wHw=(ww-vv)'*ZZZ*(ww-vv)+2*ww'*B11-vv'*B11;
B11(counter,1)=0;
beta=.5*(norm(xx-x0))^4-wHw;
tau1=Hw;
tau=tau1(t);
segma=alpha*beta+tau^2;
if segma<=.5*tau^2
     [g,G,BB,ZZZ,counter,fcounter0,MOM,x0]=RESCUE11(A,b,BB,counter
     ,ZZZ,delta,MOM,g,G,x0);
     fcounter=fcounter+fcounter0;
     rtr=1;
else
     pp=MOM(counter);
     ppp=F1(xx);
     RATIO=(pp-ppp)/(-d'*(g+G*d)-.5*d'*G*d);
     fcounter=fcounter+1;
     delta=revisedelta(RATIO,d,delta,ro);
     if t>0
          [BB,g,G,ZZZ,Z,x0,counter,lamda]=aupdate01(x0,gG,t,ww
          ,vv,d,xx,BB,ZZZ,Z,yy1,counter,pp,ppp,MOM,RATIO,g00,ss1);
          %[BB,g,G,ZZZ,Z,x0,counter,lamda]=update02(x0,g,G,t,ww,
          %vv,d,xx,BB,ZZZ,Z,yy1,counter,pp,ppp,MOM,RATIO,g00,s1);
          ss1=ss1+1;
     end
     if counter==t
          MOM(counter)=ppp;
     end
     if RATIO<0.1
          [dist1,MOVE]=table07(BB,n,counter);
          if dist1>=2*delta
          [d,delta2]=bbbBIGLAG01(MOVE,g,G,BB,ZZZ,x0,
          delta,ro,dist1,counter);
```

```
t=MOVE;
xx0=BB(:,counter)+d;
for i=1:2*n+1
     v(i)=.5*((BB(:,i)-x0)'*(BB(:,counter)-x0))^2;
end
v1=v';
wm=1;
v2=(BB(:,counter)-x0);
vv=[v1;wm;v2];
for i=1:2*n+1
     w(i)=.5*((BB(:,i)-x0)'*(xx0-x0))^2;
end
w1=w';
w2=(xx0-x0);
ww=[w1;wm;w2];
t=MOVE;
alpha=ZZZ(t,t);
B11=zeros(3*n+2,1);
B11(counter,1)=1;
Hw=ZZZ*(ww-vv)+B11;
wHw=(ww-vv)'*ZZZ*(ww-vv)+2*ww'*B11-vv'*B11;
beta=.5*(norm(xx0-x0))^4-wHw;
tau1=Hw;
B11(counter,1)=0;
tau=tau1(t);
segma=alpha*beta+tau^2;
if segma<=.5*tau^2
     [g,G,BB,ZZZ,counter,fcounter0,MOM,x0]=
     RESCUE11(A,b,BB,counter,ZZZ,delta,MOM,g,G,x0);
     fcounter=fcounter+fcounter0;
else
     pp=MOM(counter);
```

```
                                    ppp=F1(xx0);
                                    uu=uu+1;
                                    fcounter1=fcounter1+1;
                                    %[BB,g,G,ZZZ,Z,x0,counter,lamda]=aupdate01(x0,g,
                                    %G,MOVE,ww,vv,d,xx0,BB,ZZZ,Z,yy1,counter,pp,ppp,
                                    % MOM,RATIO,g00,ss1);
                                    [BB,g,G,ZZZ,Z,x0,counter,lamda]=update02(x0,g,
                                    G,MOVE,ww,vv,d,xx0,BB,ZZZ,Z,yy1,counter,pp,ppp,
                                    MOM,RATIO,g00,s1);
                               end
                               if counter==MOVE
                                    MOM(counter)=ppp;
                               end
                          else
                               max00=norm(d);
                               if delta>max00;
                                    max00=delta;
                               end
                               if (max00<=ro) & RATIO<=0
                                     ro1=ro;
                                     ro=revisedro(ro,roend);
                                     max00=0.5*ro1;
                                     if ro>max00
                                          max00=ro;
                                     end
                                     delta=max00;
                               end
                          end
                     end
               end
          else
               if norm(d)>10^-10
```

```
x1(1,mod(s1,3)+1)=norm(d);
xx1(1,mod(s1,3)+1)=abs(MOM(counter)-ppp);
s1=s1+1;
min0=x1(1);
min1=xx1(1);
for i=2:3
     if x1(i)>min0
          min0=x1(i);
     end
     if xx1(i)>min1
           min1=xx1(i);
     end
end
if( min0<=ro)&(min1<=ro^2*mincrv/8)
      ro1=ro;
      ro=revisedro(ro,roend);
      max1=.5*ro1;
      if ro>max1
            max1=ro;
      end
      delta=max1;
else
      ro1=ro;
      max1=.5*ro1;
      if ro>max1
            max1=ro;
      end
      delta=max1;
      RATIO=-1;
      [dist1,MOVE]=table07(BB,n,counter);
      if dist1>=2*delta
           [d,delta2]=bbbBIGLAG01(MOVE,g,G,BB,ZZZ,x0,
```

```
delta,ro,dist1,counter);
t=MOVE;
xx00=BB(:,counter)+d;
for i=1:2*n+1
   v(i)=.5*((BB(:,i)-x0)'*(BB(:,counter)-x0))^2;
end
v1=v';
wm=1;
v2=(BB(:,counter)-x0);
vv=[v1;wm;v2];
for i=1:2*n+1
     w(i)=.5*((BB(:,i)-x0)'*(xx00-x0))^2;
end
w1=w';
w2=(xx00-x0);
ww=[w1;wm;w2];
t=MOVE;
B11=zeros(3*n+2,1);
B11(counter,1)=1;
alpha=ZZZ(t,t);
Hw=ZZZ*(ww-vv)+B11;
wHw=(ww-vv)'*ZZZ*(ww-vv)+2*ww'*B11-vv'*B11;
B11(counter,1)=0;
beta=.5*(norm(xx00-x0))^4-wHw;
tau1=Hw;
B11(counter,1)=0;
tau=tau1(t);
segma=alpha*beta+tau^2;
if segma<=.5*tau^2
    [g,G,BB,ZZZ,counter,fcounter0,MOM,x0]=RESCUE11(A,b,BB,
    counter,ZZZ,delta,MOM,g,G,x0);
    fcounter=fcounter+fcounter0;
```

```
                    else
                        pp=MOM(counter);
                        ppp=F1(xx00);
                        uu=uu+1;
                        fcounter5=fcounter5+1;
                        [BB,g,G,ZZZ,Z,x0,counter,lamda]=aupdate01(x0,
                        g,G,t,ww,vv,d,xx00,BB,ZZZ,Z,yy1,counter,pp,ppp,MOM,
                        RATIO,g00,ss1);
                        %[BB,g,G,ZZZ,Z,x0,counter,lamda]=update02(x0,g,G,MOVE,ww,
                        %vv,d,xx,BB,ZZZ,Z,yy1,counter,pp,ppp,MOM,RATIO,g00,s1);
                    end
                    if counter==MOVE
                        MOM(counter)=ppp;
                    end
                else
                    max1=norm(d);
                    if delta>max1;
                    max1=delta;
                end
                if (max1<=ro)&(RATIO<=0)
                    ro1=ro;
                    ro=revisedro(ro,roend);
                    max1=.5*ro1;
                    if ro>max1
                        max1=ro;
                    end
                    delta=max1;
                end
            end
        end
    end
end
```

```
      e=e+1;
      if e==50000
            break
      end
   end
   fcounter=fcounter+fcounter1+fcounter5;
   if norm(d)<.5*ro
      if F1(BB(:,counter)+d)<F1(BB(:,counter))
           BB(:,counter)=BB(:,counter)+d;
      end
    end
    yy2=ones(n,1);
    if rtt==1
        x=xx;
    else
        x=BB(:,counter);
    end
    eee=norm((yy2-x),inf);
end
else
     non=length(b);
     mincrv=0;
     min101=-10^10;
     ll=1;
     ss1=1;
     s1=1;
     delta=ro;
     rtt=0;
     k1=1;
     k2=1;
     for i=1:m
         if A(i,:)*x0==b(i)
```

```
            k1=k1+1;
        else
            if A(i,:)*x0>b(i)+ro
                k2=k2+1;
            end
        end
    end
    if k2<m
        if k1<m
           x0=phase1(A,b);
        end
    end
    [g,G,BB,Z,ZZZ,yy1,counter,MOM]=initial01(x0,ro);
    xx=BB(:,counter);
    g00=g;
    g1=g;
    b0=[];
    b1=[];
    W=[];
    WW=[];
    ms=0;
    for i=1:m
       if A(i,:)*xx==b(i);
           W=[W;A(i,:)];
           b0=[b0;b(i)];
           ms=i;
       else
           WW=[WW;A(i,:)];
           b1=[b1;b(i)];
       end
    end
    [o,oo]=size(W);
```

```
WW0=[WW b1];
W0=[W b0];
if o==0
    ll=0;
end
if ll==0
     Z0Z=[];
     Y=[];
     Q=[];
     R=[];
else
     [mm,nn]=size(WW0);
     [m1,n1]=size(W0);
     W01=W0(:,1:n1-1);
     [m0,n0]=size(W01);
     [Q,R]=qr(W01');
     R1=R(1:m0,:);
     Q1=Q(:,1:m0);
     Q2=Q(:,m0+1:n0);
     Y=Q1*(inv(R1'));
     Z0Z=Q2;
     [l,l1]=size(Z0Z'*g);
     if l==0
          lamda=Y'*g;
          [p,pp]=size(lamda);
          min1=10^10;
          ss=0;
          for i=1:p
              if lamda(i)<min1
                  min1=lamda(i);
              end
              ss=i;
```

```
end
if ss==1
    f0f=W0(ss,:);
    W01=W01(2:m0,:);
    W0=W0(2:m1,:);
    WW0=[WW0;f0f];
else
    if ss==m0
        f0f=W0(ss,:);
        W01=W01(1:m0-1,:);
        W0=W0(1:m1-1,:);
        WW0=[WW0;f0f];
    else
        f0f=W0(ss,:);
        WWW=W01(1:ss-1,:);
        WWWW=W01(ss+1:m0);
        WWW1=W0(1:ss-1,:);
        WWWW1=W0(ss+1:m0);
        W01=[WWW;WWWW];
        W0=[WWW1;WWWW1];
        WW0=[WW0;f0f];
    end
end
[m0,n0]=size(W01);
[QQ,RR]=qrdelete(Q,R,ss,'col');
R1=RR(1:m0,:);
Q1=QQ(:,1:m0);
Q2=QQ(:,m0+1:n0);
Y=Q1*(inv(R1'));
R=RR;
Q=QQ;
Z0Z=Q2;
```

```
     end
end
fcounter=2*n+1;
e=1;
fcounter1=0;
fcounter2=0;
fcounter3=0;
fcounter5=0;
fcounter6=0;
uu=0;
s1=1;
x1=ro*ones(1,3);
xx1=ro*ones(1,3);
rtr=0;
ppp=0;
RATIO=.2;
d=zeros(n,1);
while ro>roend
     d0d=d;
     [d,Z0Z,Y,Q,R,min101,W0,WW0]=ccsub000333(xx,G,g,W0,WW0,Z0Z,
     Y,Q,R,delta,ll,min101,BB,x0,counter);
     st=0;
     if min101>=0
         rtt=1;
         break
     end
     dd=d;
     [ll,lll]=size(Z0Z);
     if norm(d)>=.5*ro
           xx=xx+d;
           for i=1:2*n+1
                v(i)=.5*((BB(:,i)-x0)'*(BB(:,counter)-x0))^2;
```

```
end
v1=v';
wm=1;
v2=(BB(:,counter)-x0);
vv=[v1;wm;v2];
for i=1:2*n+1
     w(i)=.5*((BB(:,i)-x0)'*(xx-x0))^2;
end
w1=w';
w2=(xx-x0);
ww=[w1;wm;w2];
B11=zeros(3*n+2,1);
B11(counter,1)=1;
t=calcul_t02(ww,vv,d,delta,x0,ro,BB,ZZZ,counter);
tt=t;
alpha=ZZZ(t,t);
Hw=ZZZ*(ww-vv)+B11;
wHw=(ww-vv)'*ZZZ*(ww-vv)+2*ww'*B11-vv'*B11;
B11(counter,1)=0;
beta=.5*(norm(xx-x0))^4-wHw;
tau1=Hw;
tau=tau1(t);
segma=alpha*beta+tau^2;
if segma<=.5*tau^2
       [g,G,BB,ZZZ,counter,fcounter0,MOM,x0]=RESCUE11(A,b,BB,
       counter,ZZZ,delta,MOM,g,G,x0);
       fcounter=fcounter+fcounter0;
       rtr=1;
else
       pp=MOM(counter);
       ppp=F1(xx);
       RATIO=(pp-ppp)/(-d'*(g+G*d)-.5*d'*G*d);
```

```
fcounter=fcounter+1;
delta=revisedelta(RATIO,d,delta,ro);
if t>0
        [BB,g,G,ZZZ,Z,x0,counter,lamda]=aupdate01(x0,g,G,t,ww
        ,vv,d,xx,BB,ZZZ,Z,yy1,counter,pp,ppp,MOM,RATIO,g00,ss1)
        ss1=ss1+1;
end
if counter==t
      MOM(counter)=ppp;
end
if RATIO<0.1
      [dist1,MOVE]=table07(BB,n,counter);
      if dist1>=2*delta
         [d,delta2]=bbbBIGLAG01(MOVE,g,G,BB,ZZZ,x0,delta,
         ro,dist1,counter);
         t=MOVE;
         xx0=BB(:,counter)+d;
         for i=1:2*n+1
             v(i)=.5*((BB(:,i)-x0)'*(BB(:,counter)-x0))^2;
         end
         v1=v';
         wm=1;
         v2=(BB(:,counter)-x0);
         vv=[v1;wm;v2];
         for i=1:2*n+1
             w(i)=.5*((BB(:,i)-x0)'*(xx0-x0))^2;
         end
         w1=w';
         w2=(xx0-x0);
         ww=[w1;wm;w2];
         t=MOVE;
         alpha=ZZZ(t,t);
```

```
    B11=zeros(3*n+2,1);
    B11(counter,1)=1;
    Hw=ZZZ*(ww-vv)+B11;
    wHw=(ww-vv)'*ZZZ*(ww-vv)+2*ww'*B11-vv'*B11;
    beta=.5*(norm(xx0-x0))^4-wHw;
    tau1=Hw;
    B11(counter,1)=0;
    tau=tau1(t);
    segma=alpha*beta+tau^2;
    if segma<=.5*tau^2
         [g,G,BB,ZZZ,counter,fcounter0,MOM,x0]=RESCUE11(A,
         BB,counter,ZZZ,delta,MOM,g,G,x0);
         fcounter=fcounter+fcounter0;
    else
         pp=MOM(counter);
         ppp=F1(xx0);
         uu=uu+1;
         fcounter1=fcounter1+1;
         [BB,g,G,ZZZ,Z,x0,counter,lamda]=aupdate01(x0,g,G,
         MOVE,ww,vv,d,xx0,BB,ZZZ,Z,yy1,counter,pp,
         ppp,MOM,RATIO,g00,ss1);
         %[BB,g,G,ZZZ,Z,x0,counter,lamda]=update02(x0,g,G,
         MOVE,ww,vv,d,xx,BB,ZZZ,Z,yy1,counter,pp,ppp,MOM,R
  end
  if counter==MOVE
         MOM(counter)=ppp;
  end
else
     max00=norm(d);
     if delta>max00;
          max00=delta;
     end
```

```
                        if (max00<=ro) & RATIO<=0
                            ro1=ro;
                            ro=revisedro(ro,roend);
                            max00=0.5*ro1;
                            if ro>max00
                                max00=ro;
                            end
                            delta=max00;
                        end
                    end
                end
            end
        else
            if norm(d)>10^-10
                x1(1,mod(s1,3)+1)=norm(d);
                xx1(1,mod(s1,3)+1)=abs(MOM(counter)-ppp);
                s1=s1+1;
                min0=x1(1);
                min1=xx1(1);
                for i=2:3
                    if x1(i)>min0
                        min0=x1(i);
                    end
                    if xx1(i)>min1
                        min1=xx1(i);
                    end
                end
                if( min0<=ro)&(min1<=ro^2*mincrv/8)
                    ro1=ro;
                    ro=revisedro(ro,roend);
                    max1=.5*ro1;
                    if ro>max1
```

```
                max1=ro;
            end
            delta=max1;
        else
            ro1=ro;
            max1=.5*ro1;
            if ro>max1
                max1=ro;
            end
            delta=max1;
            RATIO=-1;
            [dist1,MOVE]=table07(BB,n,counter);
            if dist1>=2*delta
                [d,delta2]=bbbBIGLAG01(MOVE,g,G,BB,ZZZ,x0,
                delta,ro,dist1,counter);
                t=MOVE;
                xx00=BB(:,counter)+d;
                for i=1:2*n+1
                    v(i)=.5*((BB(:,i)-x0)'*(BB(:,counter)-x0))^2;
                end
                v1=v';
                wm=1;
                v2=(BB(:,counter)-x0);
                vv=[v1;wm;v2];
                for i=1:2*n+1
                    w(i)=.5*((BB(:,i)-x0)'*(xx00-x0))^2;
                end
                w1=w';
                w2=(xx00-x0);
                ww=[w1;wm;w2];
                t=MOVE;
                B11=zeros(3*n+2,1);
```

```
B11(counter,1)=1;
alpha=ZZZ(t,t);
Hw=ZZZ*(ww-vv)+B11;
wHw=(ww-vv)'*ZZZ*(ww-vv)+2*ww'*B11-vv'*B11;
B11(counter,1)=0;
beta=.5*(norm(xx00-x0))^4-wHw;
tau1=Hw;
B11(counter,1)=0;
tau=tau1(t);
segma=alpha*beta+tau^2;
if segma<=.5*tau^2
    [g,G,BB,ZZZ,counter,fcounter0,MOM,x0]=RESCUE11(
    BB,counter,ZZZ,delta,MOM,g,G,x0);
    fcounter=fcounter+fcounter0;
else
    pp=MOM(counter);
    ppp=F1(xx00);
    uu=uu+1;
    fcounter5=fcounter5+1;
    [BB,g,G,ZZZ,Z,x0,counter,lamda]=aupdate01(x0,g,
    ,ww,vv,d,xx00,BB,ZZZ,Z,yy1,counter,pp,ppp
    ,MOM,RATIO,g00,ss1);
    %[BB,g,G,ZZZ,Z,x0,counter,lamda]=update02(x0,g,
    ,ww,vv,d,xx,BB,ZZZ,Z,yy1,counter,pp,ppp,MOM,RAT
 end
 if counter==MOVE
       MOM(counter)=ppp;
 end
else
 max1=norm(d);
 if delta>max1;
     max1=delta;
```

```
                                    end
                                    if (max1<=ro)&(RATIO<=0)
                                        ro1=ro;
                                        ro=revisedro(ro,roend);
                                        max1=.5*ro1;
                                        if ro>max1
                                            max1=ro;
                                        end
                                        delta=max1;
                                    end
                                end
                            end
                        end
                    end
                end
            end
            e=e+1;
            if e==50000
                break
            end
        end
        fcounter=fcounter+fcounter1+fcounter5;
        if norm(d)<.5*ro
            if F1(BB(:,counter)+d)<F1(BB(:,counter))
                BB(:,counter)=BB(:,counter)+d;
            end
        end
        yy2=ones(n,1);
        if rtt==1
           x=xx;
        else
           x=BB(:,counter);
        end
        eee=norm((yy2-x),inf);
```

```
end
eet=cputime;
t=eet-sst;
t=t/3600
```

B.9 Test functions

```
function yy=F1(x)
n=length(x);
y1=0;
y2=0;
y3=0;
t3=0;
yy=0;
yyy=0;
%Nocedal 2006
%yy=(x(1)-1)^2+(x(2)-5/2)^2;
%Nash and sofer 2009
%yy=.5*(x(1)-3)^2+(x(2)-2)^2;
%yy=x(1)^2+2*x(2)^2;
%Hs 01 and HS 02
%yy=100*(x(2)-x(1)^2)^2+(1-x(1))^2;
%HS 03
%yy=x(2)+10^-5*(x(2)-x(1))^2;
%Hs 04
%yy=(x(1)+1)^3/3+x(2);
%Hs 05
%yy=sin(x(1)+x(2))+(x(1)-x(2))^2-1.5*x(1)+2.5*x(2)+1;
%HS 09
%yy=sin(pi*x(1)/12)*cos(pi*x(2)/16);
%HS 21
```

```
%yy=.01*x(1)^2+x(2)^2-100;
%HS 24
%yy=(1/(27*sqrt(3)))*((x(1)-3)^2-9)*x(2)^3;
% Hs 28
%yy=(x(1)+x(2))^2+(x(2)+x(3))^2;
% Hs 35
%yy=9-8*x(1)-6*x(2)-4*x(3)+2*x(1)^2+2*x(2)^2+
%x(3)^2+2*x(1)*x(2)+2*x(1)*x(3);
%HS 36 and Hs 37
%yy=-x(1)*x(2)*x(3);
%HS 38
%yy=100*(x(2)-x(1)^2)^2+(1-x(1))^2+90*(x(4)-x(3)^2)^2
%+(1-x(3))^2+10.1*((x(2)-1)^2+(x(4)-1)^2)+19.8*(x(2)-1)*(x(4)-1);
%Hs 41
%yy=2-x(1)*x(2)*x(3);
%yy=(x(1)-x(2))^2+(x(3)-1)^2+(x(4)-1)^4+(x(5)-1)^6;
%HS 44
%yy=x(1)-x(2)-x(3)-x(1)*x(3)+x(1)*x(4)+x(2)*x(3)-x(2)*x(4);
%HS 45
%yy=2-(1/120)*x(1)*x(2)*x(3)*x(4)*x(5);
% Hs 48
%yy=(x(1)-x(2))^2+(x(2)-x(3))^2+(x(3)-x(4))^4+(x(4)-x(5))^2;
%yy=(x(1)-x(2))^2+(x(3)-1)^2+(x(4)-1)^4+(x(5)-1)^6;
% Hs 50
%yy=(x(1)-x(2))^2+(x(2)-x(3))^2+(x(3)-x(4))^4+(x(4)-x(5))^2;
% Hs 51
%yy=(x(1)-x(2))^2+(x(2)+x(3)-2)^2+(x(4)-1)^2+(x(5)-1)^2;
%yy=(4*x(1)-x(2))^2+(x(2)+x(3)-2)^2+(x(4)-1)^2+(x(5)-1)^2;
%HS 76
%yy=x(1)^2+.5*x(2)^2+x(3)^2+.5*x(4)^2-x(1)*x(3)+x(3)*
%x(4)-x(1)-3*x(2)+x(3)-x(4);
%yy=2-x(1)*x(2)*x(3);
```

```
%yy=(x(1)-1)^2+(x(2)-x(3))^2+(x(4)-x(5))^2;
%yy=-32.174*(255*log((x(1)+x(2)+x(3)+.03)/(.09*x(1)+x(2)+x(3)+.03))+
%280*log((x(2)+x(3)+.03)/(.07*x(2)+x(3)+.03))+290*log((x(3)+.03)/(.13*x(3)+.03)));
%Hs 110
% y0=1;
% for i=1:10
%     y1=y1+log((x(i)-2))^2+log(10-x(i))^2;
%     y0=y0*x(i);
% end
% yy=y1-y0^.2;
% for i=1:99
%     u=25+(-50*log(.01*i))^2/3;
%     yyy=-.01*i+exp((-1/x(1))*(u-x(2))^(x(3)));
%     yy=yy+yy^2;
% end
%yy=-32.174*(255*log((x(1)+x(2)+x(3)+.03)/(.09*x(1)+x(2)+x(3)+.03))+
%280*log((x(2)+x(3)+.03)/(.07*x(2)+x(3)+.03))+290*log((x(3)+.03)/(.13*x(3)+.03)));
%%%%%%
%the VARDIM test problem
%the initial point x0 has the components 1-i/n,i=1,1,...,n
% which takes its least at e the vector of ones
% for i=1:n
%     y1=y1+(x(i)-1)^2;
%     y2=y2+ i*(x(i)-1);
% end
% yy=y1+y2^2+y2^4;
%%%%%
%the ARWHEAD test problem
%the initial point x0 is e the vector of ones
% which takes its least value e the vector of ones except the n-th
% compoment %
for i=1:n-1
```

```
  y1=y1+((x(i)^2+x(n)^2)^2-4*x(i)+3);
end
yy=y1;
%%%%
%% The BDQRTIC test problem
%% the initial point x0 is e the vector of once
% for i=1:n-4
%   y1=y1+((x(i)^2+2*x(i+1)^2+3*x(i+2)^2+4*x(i+3)^2+5*x(n)^2)^2-4*x(i)+3);
% end
%  yy=y1;
 % The DQDRTIC function
 % the initial point x0 is [3;3;,,,,,;3]
%  for i=1:n-2
%   y1=y1+(x(i)^2+100*x(i+1)^2+100*x(i+2)^2);
% end
%  yy=y1;
% The power function
%the initial point x0 is e the vector of once
% for i=1:n
%    y1=y1+(i*x(i))^2;
% end
% yy=y1;
%the BIGGSB1 test problem
%the initial point x0  is [0;0;,,,,,;0]
% for i=1:n-1
%    y1=y1+(x(i+1)-x(i))^2+(1-x(n))^2;
% end
% yy=(x(1)-1)^2+y1;
%the CRAGGLVY test problem
%the initial point x0  is [1;2;,,,,,;2]
% for i=1:n-1
%    y1=y1+(x(i+1)-x(i))^2+(1-x(n))^2;
```

```
% end
% yy=(x(1)-1)^2+y1;
%the CHROSEN test problem
%the initial point x0  is -e
% which takes its least value of zero at e the vector of ones
% for i=1:n-1
%    y1=y1+4*(x(i)-x(i+1)^2)^2+(1-x(i+1))^2;
% end
% yy=y1;
%%%%%%%%%%%%
% The ROSENBROCK test funtion
%yy=100*(x(2)-x(1)^2)^2+(1-x(1))^2;
%%%
%%%%the trigonmetric sum of sqares test problem(TRIGSSQS)
%S=round(200*rand(2*n,n)-100);
%C=round(200*rand(2*n,n)-100);
%theta=exp((.9*randn(n,1)+.1));
%xx=2*pi*rand(n,1)-pi;
%for i=1:n
 %   yy2(i)=xx(i)/theta(i);
%end
%yy2=yy2';
%for i=1:2*n
 %   for j=1:n
  %      dd=S(i,j)*sin(xx(j))+C(i,j)*sin(xx(j));
   % end
    %b(i)=dd;
%end
%yy1=b';
% b=bb0;
% for i=1:2*n
%    dd=0;
```

```
%    for j=1:n
%        dd=dd+S(i,j)*sin(theta(j)*x(j))+C(i,j)*sin(theta(j)*x(j));
%    end
%     yy=yy+(b(i)-dd)^2;
% end
% yy=(x(1)-x(2)+.2)^2+10^-4*(x(1)+x(2)+77)^2;
%%%%%%%
%the PENALTY1 test problem
%the initial point x0  is e
% which takes its
% for i=1:n
%     y1=y1+(x(i)-1)^2;
%     y2=y2+x(i)^2;
% end
% yy=(1/16)+10^(-5)*y1+y2^2-.5*y2;
%%%%%%%%
% the PENALTY2 test problem
%
% for i=1:n
%     y1=y1+(n-i+1)*x(i)^2;
%     if i>=2
%         y2=y2+(exp(x(i-1)/10)+exp(x(i)/10)-exp((i-1)/10)-
%exp(i/10))^2+(exp(x(i)/10)-exp(-1/10))^2;
%     end
% end
% y3=1-2*y1+y1^2;
% yy=y3+y2+(x(1)-.2)^2;
% the PENALTY3 test problem
% y4=0;
% for i=1:n-2
%     y1=y1+(x(i)+2*x(i+1)+10*x(i+2)-1)^2;
%     y2=y2+(2*x(i)+x(i+1)-3)^2;
```

```
%     y3=y3+x(i)^2-n;
% end
% for i=1:n/2
%     y4=y4+(x(i)-1)^2;
% end
% y3=y3^2;
% R=y1;
% S=y2;
% yy=10^-3*(1+R*exp(x(n))+S*exp(x(n-1))+R*S)+y3+y4;
%the SPHRPTS test problem
% for k=1:n/2
%     BB(:,k)=[x(2*k-1);x(2*k)];
%     %BB(:,k)=[cos(x(2*k-1))*cos(x(2*k));sin(x(2*k-1))*cos(x(2*k));sin(x(2*k))];
% end
% BB;
% for k=2:n/2
%     for l=1:k-1
%         yyy=norm(BB(:,l)-BB(:,k))^-1;
%         if yyy>10^3
%             yyy=10^3;
%         end
%         yy=yy+yyy;
%     end
% end
%%%The Rosenbrock's function
%yy=100*(x(2)-x(1)^2)^2+(1-x(1))^2;
%%%% the Singular function
%yy=(x(1)+10*(x(2)))^2+5*(x(3)-x(4))^2+(x(2)-2*x(3))^4+10*(x(1)-x(4))^4;
```